halo-dev / halo Public

Code | Issues 214 | Pull requests 7 | Discussions | Actions | Projects | ...

v1.4.17

halo / src / main / java / run / halo / app / service / impl / **PostServiceImpl.java** / Jump to

fuzui Fix number of comment in detail of post and sheet and add it into det... ... ✓ History

16 contributors +4

1055 lines (859 sloc) | 39.3 KB

```java
package run.halo.app.service.impl;

import static org.springframework.data.domain.Sort.Direction.DESC;
import static run.halo.app.model.support.HaloConst.URL_SEPARATOR;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import javax.persistence.criteria.Subquery;
import javax.validation.constraints.NotNull;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.lang.NonNull;
import org.springframework.lang.Nullable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.Assert;
import org.springframework.util.CollectionUtils;
import run.halo.app.event.logger.LogEvent;
import run.halo.app.event.post.PostVisitEvent;
import run.halo.app.exception.NotFoundException;
import run.halo.app.model.dto.post.BasePostMinimalDTO;
import run.halo.app.model.dto.post.BasePostSimpleDTO;
import run.halo.app.model.entity.Category;
import run.halo.app.model.entity.Post;
import run.halo.app.model.entity.PostCategory;
import run.halo.app.model.entity.PostComment;
import run.halo.app.model.entity.PostMeta;
import run.halo.app.model.entity.PostTag;
import run.halo.app.model.entity.Tag;
import run.halo.app.model.enums.CommentStatus;
import run.halo.app.model.enums.LogType;
import run.halo.app.model.enums.PostPermalinkType;
import run.halo.app.model.enums.PostStatus;
import run.halo.app.model.params.PostParam;
import run.halo.app.model.params.PostQuery;
import run.halo.app.model.properties.PostProperties;
import run.halo.app.model.vo.ArchiveMonthVO;
import run.halo.app.model.vo.ArchiveYearVO;
import run.halo.app.model.vo.PostDetailVO;
import run.halo.app.model.vo.PostListVO;
import run.halo.app.model.vo.PostMarkdownVO;
import run.halo.app.repository.PostRepository;
import run.halo.app.repository.base.BasePostRepository;
import run.halo.app.service.AuthorizationService;
import run.halo.app.service.CategoryService;
import run.halo.app.service.OptionService;
import run.halo.app.service.PostCategoryService;
import run.halo.app.service.PostCommentService;
import run.halo.app.service.PostMetaService;
import run.halo.app.service.PostService;
import run.halo.app.service.PostTagService;
import run.halo.app.service.TagService;
import run.halo.app.utils.DateUtils;
import run.halo.app.utils.HaloUtils;
import run.halo.app.utils.MarkdownUtils;
import run.halo.app.utils.ServiceUtils;
import run.halo.app.utils.SlugUtils;

/**
```

```java
 * Post service implementation.
 *
 * @author johnniang
 * @author ryanwang
 * @author guqing
 * @author evanwang
 * @author coor.top
 * @author Raremaa
 * @date 2019-03-14
 */
@Slf4j
@Service
public class PostServiceImpl extends BasePostServiceImpl<Post> implements PostService {

    private final PostRepository postRepository;

    private final TagService tagService;

    private final CategoryService categoryService;

    private final PostTagService postTagService;

    private final PostCategoryService postCategoryService;

    private final PostCommentService postCommentService;

    private final ApplicationEventPublisher eventPublisher;

    private final PostMetaService postMetaService;

    private final OptionService optionService;

    private final AuthorizationService authorizationService;

    public PostServiceImpl(BasePostRepository<Post> basePostRepository,
        OptionService optionService,
        PostRepository postRepository,
        TagService tagService,
        CategoryService categoryService,
        PostTagService postTagService,
        PostCategoryService postCategoryService,
        PostCommentService postCommentService,
        ApplicationEventPublisher eventPublisher,
        PostMetaService postMetaService,
        AuthorizationService authorizationService) {
        super(basePostRepository, optionService);
        this.postRepository = postRepository;
        this.tagService = tagService;
        this.categoryService = categoryService;
        this.postTagService = postTagService;
        this.postCategoryService = postCategoryService;
        this.postCommentService = postCommentService;
        this.eventPublisher = eventPublisher;
        this.postMetaService = postMetaService;
        this.optionService = optionService;
        this.authorizationService = authorizationService;
    }

    @Override
    public Page<Post> pageBy(PostQuery postQuery, Pageable pageable) {
        Assert.notNull(postQuery, "Post query must not be null");
        Assert.notNull(pageable, "Page info must not be null");

        // Build specification and find all
        return postRepository.findAll(buildSpecByQuery(postQuery), pageable);
    }

    @Override
    public Page<Post> pageBy(String keyword, Pageable pageable) {
        Assert.notNull(keyword, "keyword must not be null");
        Assert.notNull(pageable, "Page info must not be null");

        PostQuery postQuery = new PostQuery();
        postQuery.setKeyword(keyword);
        postQuery.setStatus(PostStatus.PUBLISHED);

        // Build specification and find all
        return postRepository.findAll(buildSpecByQuery(postQuery), pageable);
    }

    @Override
    @Transactional
    public PostDetailVO createBy(Post postToCreate, Set<Integer> tagIds, Set<Integer> categoryIds,
        Set<PostMeta> metas, boolean autoSave) {
        PostDetailVO createdPost = createOrUpdate(postToCreate, tagIds, categoryIds, metas);
        if (!autoSave) {
            // Log the creation
            LogEvent logEvent = new LogEvent(this, createdPost.getId().toString(),
                LogType.POST_PUBLISHED, createdPost.getTitle());
            eventPublisher.publishEvent(logEvent);
        }
        return createdPost;
    }

    @Override
    public PostDetailVO createBy(Post postToCreate, Set<Integer> tagIds, Set<Integer> categoryIds,
        boolean autoSave) {
        PostDetailVO createdPost = createOrUpdate(postToCreate, tagIds, categoryIds, null);
```

```java
            if (!autoSave) {
                // Log the creation
                LogEvent logEvent = new LogEvent(this, createdPost.getId().toString(),
                    LogType.POST_PUBLISHED, createdPost.getTitle());
                eventPublisher.publishEvent(logEvent);
            }
            return createdPost;
        }

        @Override
        @Transactional
        public PostDetailVO updateBy(Post postToUpdate, Set<Integer> tagIds, Set<Integer> categoryIds,
            Set<PostMeta> metas, boolean autoSave) {
            // Set edit time
            postToUpdate.setEditTime(DateUtils.now());
            PostDetailVO updatedPost = createOrUpdate(postToUpdate, tagIds, categoryIds, metas);
            if (!autoSave) {
                // Log the creation
                LogEvent logEvent = new LogEvent(this, updatedPost.getId().toString(),
                    LogType.POST_EDITED, updatedPost.getTitle());
                eventPublisher.publishEvent(logEvent);
            }
            return updatedPost;
        }

        @Override
        public Post getBy(PostStatus status, String slug) {
            return super.getBy(status, slug);
        }

        @Override
        public Post getBy(Integer year, Integer month, String slug) {
            Assert.notNull(year, "Post create year must not be null");
            Assert.notNull(month, "Post create month must not be null");
            Assert.notNull(slug, "Post slug must not be null");

            Optional<Post> postOptional = postRepository.findBy(year, month, slug);

            return postOptional
                .orElseThrow(() -> new NotFoundException("查询不到该文章的信息").setErrorData(slug));
        }

        @NonNull
        @Override
        public Post getBy(@NonNull Integer year, @NonNull String slug) {
            Assert.notNull(year, "Post create year must not be null");
            Assert.notNull(slug, "Post slug must not be null");

            Optional<Post> postOptional = postRepository.findBy(year, slug);

            return postOptional
                .orElseThrow(() -> new NotFoundException("查询不到该文章的信息").setErrorData(slug));
        }

        @Override
        public Post getBy(Integer year, Integer month, String slug, PostStatus status) {
            Assert.notNull(year, "Post create year must not be null");
            Assert.notNull(month, "Post create month must not be null");
            Assert.notNull(slug, "Post slug must not be null");
            Assert.notNull(status, "Post status must not be null");

            Optional<Post> postOptional = postRepository.findBy(year, month, slug, status);

            return postOptional
                .orElseThrow(() -> new NotFoundException("查询不到该文章的信息").setErrorData(slug));
        }

        @Override
        public Post getBy(Integer year, Integer month, Integer day, String slug) {
            Assert.notNull(year, "Post create year must not be null");
            Assert.notNull(month, "Post create month must not be null");
            Assert.notNull(day, "Post create day must not be null");
            Assert.notNull(slug, "Post slug must not be null");

            Optional<Post> postOptional = postRepository.findBy(year, month, day, slug);

            return postOptional
                .orElseThrow(() -> new NotFoundException("查询不到该文章的信息").setErrorData(slug));
        }

        @Override
        public Post getBy(Integer year, Integer month, Integer day, String slug, PostStatus status) {
            Assert.notNull(year, "Post create year must not be null");
            Assert.notNull(month, "Post create month must not be null");
            Assert.notNull(day, "Post create day must not be null");
            Assert.notNull(slug, "Post slug must not be null");
            Assert.notNull(status, "Post status must not be null");

            Optional<Post> postOptional = postRepository.findBy(year, month, day, slug, status);

            return postOptional
                .orElseThrow(() -> new NotFoundException("查询不到该文章的信息").setErrorData(slug));
        }

        @Override
        public List<Post> removeByIds(Collection<Integer> ids) {
            if (CollectionUtils.isEmpty(ids)) {
                return Collections.emptyList();
            }
            return ids.stream().map(this::removeById).collect(Collectors.toList());
        }

        @Override
        public Post getBySlug(String slug) {
            return super.getBySlug(slug);
        }

        @Override
        public List<ArchiveYearVO> listYearArchives() {
            // Get all posts
            List<Post> posts = postRepository
                .findAllByStatus(PostStatus.PUBLISHED, Sort.by(DESC, "createTime"));

            return convertToYearArchives(posts);
        }

        @Override
        public List<ArchiveMonthVO> listMonthArchives() {
            // Get all posts
            List<Post> posts = postRepository
                .findAllByStatus(PostStatus.PUBLISHED, Sort.by(DESC, "createTime"));

            return convertToMonthArchives(posts);
        }

        @Override
        public List<ArchiveYearVO> convertToYearArchives(List<Post> posts) {
            Map<Integer, List<Post>> yearPostMap = new HashMap<>(8);

            posts.forEach(post -> {
                Calendar calendar = DateUtils.convertTo(post.getCreateTime());
                yearPostMap.computeIfAbsent(calendar.get(Calendar.YEAR), year -> new LinkedList<>())
                    .add(post);
            });

            List<ArchiveYearVO> archives = new LinkedList<>();

            yearPostMap.forEach((year, postList) -> {
                // Build archive
                ArchiveYearVO archive = new ArchiveYearVO();
                archive.setYear(year);
                archive.setPosts(convertToListVo(postList));

                // Add archive
                archives.add(archive);
            });

            // Sort this list
            archives.sort(new ArchiveYearVO.ArchiveComparator());

            return archives;
        }

        @Override
        public List<ArchiveMonthVO> convertToMonthArchives(List<Post> posts) {

            Map<Integer, Map<Integer, List<Post>>> yearMonthPostMap = new HashMap<>(8);

            posts.forEach(post -> {
                Calendar calendar = DateUtils.convertTo(post.getCreateTime());

                yearMonthPostMap.computeIfAbsent(calendar.get(Calendar.YEAR), year -> new HashMap<>())
                    .computeIfAbsent(calendar.get(Calendar.MONTH) + 1,
                        month -> new LinkedList<>())
                    .add(post);
            });

            List<ArchiveMonthVO> archives = new LinkedList<>();

            yearMonthPostMap.forEach((year, monthPostMap) ->
                monthPostMap.forEach((month, postList) -> {
                    ArchiveMonthVO archive = new ArchiveMonthVO();
                    archive.setYear(year);
                    archive.setMonth(month);
                    archive.setPosts(convertToListVo(postList));

                    archives.add(archive);
                }));

            // Sort this list
            archives.sort(new ArchiveMonthVO.ArchiveComparator());

            return archives;
        }

        @Override
        public PostDetailVO importMarkdown(String markdown, String filename) {
            Assert.notNull(markdown, "Markdown document must not be null");

            // Gets frontMatter
            Map<String, List<String>> frontMatter = MarkdownUtils.getFrontMatter(markdown);
            // remove frontMatter
            markdown = MarkdownUtils.removeFrontMatter(markdown);

            PostParam post = new PostParam();
            post.setStatus(null);

            List<String> elementValue;

            Set<Integer> tagIds = new HashSet<>();

            Set<Integer> categoryIds = new HashSet<>();

            if (frontMatter.size() > 0) {
                for (String key : frontMatter.keySet()) {
                    elementValue = frontMatter.get(key);
                    for (String ele : elementValue) {
                        ele = HaloUtils.strip(ele, "[", "]");
                        ele = StringUtils.strip(ele, "\"");
                        ele = StringUtils.strip(ele, "\'");
                        if ("".equals(ele)) {
                            continue;
                        }
                        switch (key) {
                            case "title":
                                post.setTitle(ele);
                                break;
                            case "date":
                                post.setCreateTime(DateUtils.parseDate(ele));
                                break;
                            case "permalink":
                                post.setSlug(ele);
                                break;
                            case "thumbnail":
                                post.setThumbnail(ele);
                                break;
                            case "status":
                                post.setStatus(PostStatus.valueOf(ele));
                                break;
                            case "comments":
                                post.setDisallowComment(Boolean.parseBoolean(ele));
                                break;
                            case "tags":
                                Tag tag;
                                for (String tagName : ele.split(",")) {
                                    tagName = tagName.trim();
                                    tagName = StringUtils.strip(tagName, "\"");
                                    tagName = StringUtils.strip(tagName, "\'");
                                    tag = tagService.getByName(tagName);
                                    String slug = SlugUtils.slug(tagName);
                                    if (null == tag) {
                                        tag = tagService.getBySlug(slug);
                                    }
                                    if (null == tag) {
                                        tag = new Tag();
```

```java
                                    tag.setName(tagName);
                                    tag.setSlug(slug);
                                    tag = tagService.create(tag);
                                }
                                tagIds.add(tag.getId());
                            }
                            break;
                        case "categories":
                            Integer lastCategoryId = null;
                            for (String categoryName : ele.split(",")) {
                                categoryName = categoryName.trim();
                                categoryName = StringUtils.strip(categoryName, "\"");
                                categoryName = StringUtils.strip(categoryName, "\'");
                                Category category = categoryService.getByName(categoryName);
                                if (null == category) {
                                    category = new Category();
                                    category.setName(categoryName);
                                    category.setSlug(SlugUtils.slug(categoryName));
                                    category.setDescription(categoryName);
                                    if (lastCategoryId != null) {
                                        category.setParentId(lastCategoryId);
                                    }
                                    category = categoryService.create(category);
                                }
                                lastCategoryId = category.getId();
                                categoryIds.add(lastCategoryId);
                            }
                            break;
                        default:
                            break;
                    }
                }
            }
        }

        if (null == post.getStatus()) {
            post.setStatus(PostStatus.PUBLISHED);
        }

        if (StringUtils.isEmpty(post.getTitle())) {
            post.setTitle(filename);
        }

        if (StringUtils.isEmpty(post.getSlug())) {
            post.setSlug(SlugUtils.slug(post.getTitle()));
        }

        post.setOriginalContent(markdown);

        return createBy(post.convertTo(), tagIds, categoryIds, false);
    }

    @Override
    public String exportMarkdown(Integer id) {
        Assert.notNull(id, "Post id must not be null");
        Post post = getById(id);
        return exportMarkdown(post);
    }

    @Override
    public String exportMarkdown(Post post) {
        Assert.notNull(post, "Post must not be null");

        StringBuilder content = new StringBuilder("---\n");

        content.append("type: ").append("post").append("\n");
        content.append("title: ").append(post.getTitle()).append("\n");
        content.append("permalink: ").append(post.getSlug()).append("\n");
        content.append("thumbnail: ").append(post.getThumbnail()).append("\n");
        content.append("status: ").append(post.getStatus()).append("\n");
        content.append("date: ").append(post.getCreateTime()).append("\n");
        content.append("updated: ").append(post.getEditTime()).append("\n");
        content.append("comments: ").append(!post.getDisallowComment()).append("\n");

        List<Tag> tags = postTagService.listTagsBy(post.getId());

        if (tags.size() > 0) {
            content.append("tags:").append("\n");
            for (Tag tag : tags) {
                content.append("  - ").append(tag.getName()).append("\n");
            }
        }

        List<Category> categories = postCategoryService.listCategoriesBy(post.getId());

        if (categories.size() > 0) {
            content.append("categories:").append("\n");
            for (Category category : categories) {
                content.append("  - ").append(category.getName()).append("\n");
            }
        }

        List<PostMeta> metas = postMetaService.listBy(post.getId());

        if (metas.size() > 0) {
            content.append("metas:").append("\n");
            for (PostMeta postMeta : metas) {
                content.append("  - ").append(postMeta.getKey()).append(" :  ")
```

```java
                        .append(postMeta.getValue()).append("\n");
                }
            }

            content.append("---\n\n");
            content.append(post.getOriginalContent());
            return content.toString();
        }

        @Override
        public PostDetailVO convertToDetailVo(Post post) {
            return convertToDetailVo(post, false);
        }

        @Override
        public PostDetailVO convertToDetailVo(Post post, boolean queryEncryptCategory) {
            // List tags
            List<Tag> tags = postTagService.listTagsBy(post.getId());
            // List categories
            List<Category> categories = postCategoryService
                .listCategoriesBy(post.getId(), queryEncryptCategory);
            // List metas
            List<PostMeta> metas = postMetaService.listBy(post.getId());
            // Convert to detail vo
            return convertTo(post, tags, categories, metas);
        }

        @Override
        public Page<PostDetailVO> convertToDetailVo(Page<Post> postPage) {
            Assert.notNull(postPage, "Post page must not be null");
            return postPage.map(this::convertToDetailVo);
        }

        @Override
        public Post removeById(Integer postId) {
            Assert.notNull(postId, "Post id must not be null");

            log.debug("Removing post: [{}]", postId);

            // Remove post tags
            List<PostTag> postTags = postTagService.removeByPostId(postId);

            log.debug("Removed post tags: [{}]", postTags);

            // Remove post categories
            List<PostCategory> postCategories = postCategoryService.removeByPostId(postId);

            log.debug("Removed post categories: [{}]", postCategories);

            // Remove metas
            List<PostMeta> metas = postMetaService.removeByPostId(postId);
            log.debug("Removed post metas: [{}]", metas);

            // Remove post comments
            List<PostComment> postComments = postCommentService.removeByPostId(postId);
            log.debug("Removed post comments: [{}]", postComments);

            Post deletedPost = super.removeById(postId);

            // Log it
            eventPublisher.publishEvent(new LogEvent(this, postId.toString(), LogType.POST_DELETED,
                deletedPost.getTitle()));

            return deletedPost;
        }

        @Override
        public Page<PostListVO> convertToListVo(Page<Post> postPage) {
            return convertToListVo(postPage, false);
        }

        @Override
        public Page<PostListVO> convertToListVo(Page<Post> postPage, boolean queryEncryptCategory) {
            Assert.notNull(postPage, "Post page must not be null");

            List<Post> posts = postPage.getContent();

            Set<Integer> postIds = ServiceUtils.fetchProperty(posts, Post::getId);

            // Get tag list map
            Map<Integer, List<Tag>> tagListMap = postTagService.listTagListMapBy(postIds);

            // Get category list map
            Map<Integer, List<Category>> categoryListMap = postCategoryService
                .listCategoryListMap(postIds, queryEncryptCategory);

            // Get comment count
            Map<Integer, Long> commentCountMap = postCommentService.countByStatusAndPostIds(
                CommentStatus.PUBLISHED, postIds);

            // Get post meta list map
            Map<Integer, List<PostMeta>> postMetaListMap = postMetaService.listPostMetaAsMap(postIds);

            return postPage.map(post -> {
                PostListVO postListVO = new PostListVO().convertFrom(post);

                if (StringUtils.isBlank(postListVO.getSummary())) {
                    postListVO.setSummary(generateSummary(post.getFormatContent()));
                }

                Optional.ofNullable(tagListMap.get(post.getId())).orElseGet(LinkedList::new);

                // Set tags
                postListVO.setTags(Optional.ofNullable(tagListMap.get(post.getId()))
                    .orElseGet(LinkedList::new)
                    .stream()
                    .filter(Objects::nonNull)
                    .map(tagService::convertTo)
                    .collect(Collectors.toList()));

                // Set categories
                postListVO.setCategories(Optional.ofNullable(categoryListMap.get(post.getId()))
                    .orElseGet(LinkedList::new)
                    .stream()
                    .filter(Objects::nonNull)
                    .map(categoryService::convertTo)
                    .collect(Collectors.toList()));

                // Set post metas
                List<PostMeta> metas = Optional.ofNullable(postMetaListMap.get(post.getId()))
                    .orElseGet(LinkedList::new);
                postListVO.setMetas(postMetaService.convertToMap(metas));

                // Set comment count
                postListVO.setCommentCount(commentCountMap.getOrDefault(post.getId(), 0L));

                postListVO.setFullPath(buildFullPath(post));

                return postListVO;
            });
        }

        @Override
        public List<PostListVO> convertToListVo(List<Post> posts) {
            return convertToListVo(posts, false);
        }

        @Override
        public List<PostListVO> convertToListVo(List<Post> posts, boolean queryEncryptCategory) {
            Assert.notNull(posts, "Post page must not be null");

            Set<Integer> postIds = ServiceUtils.fetchProperty(posts, Post::getId);

            // Get tag list map
            Map<Integer, List<Tag>> tagListMap = postTagService.listTagListMapBy(postIds);

            // Get category list map
            Map<Integer, List<Category>> categoryListMap = postCategoryService
                .listCategoryListMap(postIds, queryEncryptCategory);

            // Get comment count
            Map<Integer, Long> commentCountMap =
                postCommentService.countByStatusAndPostIds(CommentStatus.PUBLISHED, postIds);

            // Get post meta list map
            Map<Integer, List<PostMeta>> postMetaListMap = postMetaService.listPostMetaAsMap(postIds);

            return posts.stream().map(post -> {
                PostListVO postListVO = new PostListVO().convertFrom(post);

                if (StringUtils.isBlank(postListVO.getSummary())) {
                    postListVO.setSummary(generateSummary(post.getFormatContent()));
                }

                Optional.ofNullable(tagListMap.get(post.getId())).orElseGet(LinkedList::new);

                // Set tags
                postListVO.setTags(Optional.ofNullable(tagListMap.get(post.getId()))
                    .orElseGet(LinkedList::new)
                    .stream()
                    .filter(Objects::nonNull)
                    .map(tagService::convertTo)
                    .collect(Collectors.toList()));

                // Set categories
                postListVO.setCategories(Optional.ofNullable(categoryListMap.get(post.getId()))
                    .orElseGet(LinkedList::new)
                    .stream()
                    .filter(Objects::nonNull)
                    .map(categoryService::convertTo)
                    .collect(Collectors.toList()));

                // Set post metas
                List<PostMeta> metas = Optional.ofNullable(postMetaListMap.get(post.getId()))
                    .orElseGet(LinkedList::new);
                postListVO.setMetas(postMetaService.convertToMap(metas));

                // Set comment count
                postListVO.setCommentCount(commentCountMap.getOrDefault(post.getId(), 0L));

                postListVO.setFullPath(buildFullPath(post));

                return postListVO;
            }).collect(Collectors.toList());
        }

        @Override
        public BasePostMinimalDTO convertToMinimal(Post post) {
            Assert.notNull(post, "Post must not be null");
            BasePostMinimalDTO basePostMinimalDTO = new BasePostMinimalDTO().convertFrom(post);

            basePostMinimalDTO.setFullPath(buildFullPath(post));

            return basePostMinimalDTO;
        }

        @Override
        public List<BasePostMinimalDTO> convertToMinimal(List<Post> posts) {
            if (CollectionUtils.isEmpty(posts)) {
                return Collections.emptyList();
            }

            return posts.stream()
                .map(this::convertToMinimal)
                .collect(Collectors.toList());
        }

        @Override
        public BasePostSimpleDTO convertToSimple(Post post) {
            Assert.notNull(post, "Post must not be null");

            BasePostSimpleDTO basePostSimpleDTO = new BasePostSimpleDTO().convertFrom(post);

            // Set summary
            if (StringUtils.isBlank(basePostSimpleDTO.getSummary())) {
                basePostSimpleDTO.setSummary(generateSummary(post.getFormatContent()));
            }

            basePostSimpleDTO.setFullPath(buildFullPath(post));

            return basePostSimpleDTO;
        }

        /**
         * Converts to post detail vo.
         *
         * @param post post must not be null
         * @param tags tags
         * @param categories categories
         * @param postMetaList postMetaList
         * @return post detail vo
         */
        @NonNull
        private PostDetailVO convertTo(@NonNull Post post, @Nullable List<Tag> tags,
            @Nullable List<Category> categories, List<PostMeta> postMetaList) {
```

```java
        Assert.notNull(post, "Post must not be null");

        // Convert to base detail vo
        PostDetailVO postDetailVO = new PostDetailVO().convertFrom(post);

        if (StringUtils.isBlank(postDetailVO.getSummary())) {
            postDetailVO.setSummary(generateSummary(post.getFormatContent()));
        }

        // Extract ids
        Set<Integer> tagIds = ServiceUtils.fetchProperty(tags, Tag::getId);
        Set<Integer> categoryIds = ServiceUtils.fetchProperty(categories, Category::getId);
        Set<Long> metaIds = ServiceUtils.fetchProperty(postMetaList, PostMeta::getId);

        // Get post tag ids
        postDetailVO.setTagIds(tagIds);
        postDetailVO.setTags(tagService.convertTo(tags));

        // Get post category ids
        postDetailVO.setCategoryIds(categoryIds);
        postDetailVO.setCategories(categoryService.convertTo(categories));

        // Get post meta ids
        postDetailVO.setMetaIds(metaIds);
        postDetailVO.setMetas(postMetaService.convertTo(postMetaList));

        postDetailVO.setCommentCount(postCommentService.countByStatusAndPostId(
            CommentStatus.PUBLISHED, post.getId()));

        postDetailVO.setFullPath(buildFullPath(post));

        return postDetailVO;
    }

    /**
     * Build specification by post query.
     *
     * @param postQuery post query must not be null
     * @return a post specification
     */
    @NonNull
    private Specification<Post> buildSpecByQuery(@NonNull PostQuery postQuery) {
        Assert.notNull(postQuery, "Post query must not be null");

        return (root, query, criteriaBuilder) -> {
            List<Predicate> predicates = new LinkedList<>();

            if (postQuery.getStatus() != null) {
                predicates.add(criteriaBuilder.equal(root.get("status"), postQuery.getStatus()));
            }

            if (postQuery.getCategoryId() != null) {
                Subquery<Post> postSubquery = query.subquery(Post.class);
                Root<PostCategory> postCategoryRoot = postSubquery.from(PostCategory.class);
                postSubquery.select(postCategoryRoot.get("postId"));
                postSubquery.where(
                    criteriaBuilder.equal(root.get("id"), postCategoryRoot.get("postId")),
                    criteriaBuilder
                        .equal(postCategoryRoot.get("categoryId"), postQuery.getCategoryId()));
                predicates.add(criteriaBuilder.exists(postSubquery));
            }

            if (postQuery.getKeyword() != null) {
                // Format like condition
                String likeCondition = String
                    .format("%%%s%%", StringUtils.strip(postQuery.getKeyword()));

                // Build like predicate
                Predicate titleLike = criteriaBuilder.like(root.get("title"), likeCondition);
                Predicate originalContentLike = criteriaBuilder
                    .like(root.get("originalContent"), likeCondition);

                predicates.add(criteriaBuilder.or(titleLike, originalContentLike));
            }

            return query.where(predicates.toArray(new Predicate[0])).getRestriction();
        };
    }

    private PostDetailVO createOrUpdate(@NonNull Post post, Set<Integer> tagIds,
        Set<Integer> categoryIds, Set<PostMeta> metas) {
        Assert.notNull(post, "Post param must not be null");

        // Create or update post
        Boolean needEncrypt = Optional.ofNullable(categoryIds)
            .filter(HaloUtils::isNotEmpty)
            .map(categoryIdSet -> {
                for (Integer categoryId : categoryIdSet) {
                    if (categoryService.categoryHasEncrypt(categoryId)) {
                        return true;
                    }
                }
                return false;
            }).orElse(Boolean.FALSE);

        // if password is not empty or parent category has encrypt, change status to intimate
        if (post.getStatus() != PostStatus.DRAFT
            && (StringUtils.isNotEmpty(post.getPassword()) || needEncrypt)
        ) {
            post.setStatus(PostStatus.INTIMATE);
        }

        post = super.createOrUpdateBy(post);

        postTagService.removeByPostId(post.getId());

        postCategoryService.removeByPostId(post.getId());

        // List all tags
        List<Tag> tags = tagService.listAllByIds(tagIds);

        // List all categories
        List<Category> categories = categoryService.listAllByIds(categoryIds, true);

        // Create post tags
        List<PostTag> postTags = postTagService.mergeOrCreateByIfAbsent(post.getId(),
            ServiceUtils.fetchProperty(tags, Tag::getId));

        log.debug("Created post tags: [{}]", postTags);

        // Create post categories
        List<PostCategory> postCategories =
            postCategoryService.mergeOrCreateByIfAbsent(post.getId(),
                ServiceUtils.fetchProperty(categories, Category::getId));

        log.debug("Created post categories: [{}]", postCategories);

        // Create post meta data
        List<PostMeta> postMetaList = postMetaService
            .createOrUpdateByPostId(post.getId(), metas);
        log.debug("Created post metas: [{}]", postMetaList);

        // Remove authorization every time an post is created or updated.
        authorizationService.deletePostAuthorization(post.getId());

        // Convert to post detail vo
        return convertTo(post, tags, categories, postMetaList);
    }

    @Override
    @Transactional
    public Post updateStatus(PostStatus status, Integer postId) {
        super.updateStatus(status, postId);
        if (PostStatus.PUBLISHED.equals(status)) {
            // When the update status is published, it is necessary to determine whether
            // the post status should be converted to a intimate post
            categoryService.refreshPostStatus(Collections.singletonList(postId));
        }
        return getById(postId);
    }

    @Override
    @Transactional
    public List<Post> updateStatusByIds(List<Integer> ids, PostStatus status) {
        if (CollectionUtils.isEmpty(ids)) {
            return Collections.emptyList();
        }
        return ids.stream().map(id -> updateStatus(status, id)).collect(Collectors.toList());
    }

    @Override
    public void publishVisitEvent(Integer postId) {
        eventPublisher.publishEvent(new PostVisitEvent(this, postId));
    }

    @Override
    public @NotNull Sort getPostDefaultSort() {
        String indexSort = optionService.getByPropertyOfNonNull(PostProperties.INDEX_SORT)
            .toString();
        return Sort.by(DESC, "topPriority").and(Sort.by(DESC, indexSort).and(Sort.by(DESC, "id")))
    }

    @Override
    public List<PostMarkdownVO> listPostMarkdowns() {
        List<Post> allPostList = listAll();
        List<PostMarkdownVO> result = new ArrayList(allPostList.size());
        for (int i = 0; i < allPostList.size(); i++) {
            Post post = allPostList.get(i);
            result.add(convertToPostMarkdownVo(post));
        }
        return result;
    }

    private PostMarkdownVO convertToPostMarkdownVo(Post post) {
        PostMarkdownVO postMarkdownVO = new PostMarkdownVO();

        StringBuilder frontMatter = new StringBuilder("---\n");
        frontMatter.append("title: ").append(post.getTitle()).append("\n");
        frontMatter.append("date: ").append(post.getCreateTime()).append("\n");
        frontMatter.append("updated: ").append(post.getUpdateTime()).append("\n");

        //set fullPath
        frontMatter.append("url: ").append(buildFullPath(post)).append("\n");

        //set category
        List<Category> categories = postCategoryService.listCategoriesBy(post.getId());
        StringBuilder categoryContent = new StringBuilder();
        for (int i = 0; i < categories.size(); i++) {
            Category category = categories.get(i);
            String categoryName = category.getName();
            if (i == 0) {
                categoryContent.append(categoryName);
            } else {
                categoryContent.append(" | ").append(categoryName);
            }
        }
        frontMatter.append("categories: ").append(categoryContent.toString()).append("\n");

        //set tags
        List<Tag> tags = postTagService.listTagsBy(post.getId());
        StringBuilder tagContent = new StringBuilder();
        for (int i = 0; i < tags.size(); i++) {
            Tag tag = tags.get(i);
            String tagName = tag.getName();
            if (i == 0) {
                tagContent.append(tagName);
            } else {
                tagContent.append(" | ").append(tagName);
            }
        }
        frontMatter.append("tags: ").append(tagContent.toString()).append("\n");

        frontMatter.append("---\n");
        postMarkdownVO.setFrontMatter(frontMatter.toString());
        postMarkdownVO.setOriginalContent(post.getOriginalContent());
        postMarkdownVO.setTitle(post.getTitle());
        postMarkdownVO.setSlug(post.getSlug());
        return postMarkdownVO;
    }

    private String buildFullPath(Post post) {

        PostPermalinkType permalinkType = optionService.getPostPermalinkType();

        String pathSuffix = optionService.getPathSuffix();

        String archivesPrefix = optionService.getArchivesPrefix();

        int month = DateUtils.month(post.getCreateTime()) + 1;

        String monthString = month < 10 ? "0" + month : String.valueOf(month);

        int day = DateUtils.dayOfMonth(post.getCreateTime());

        String dayString = day < 10 ? "0" + day : String.valueOf(day);

        StringBuilder fullPath = new StringBuilder();

        if (optionService.isEnabledAbsolutePath()) {
            fullPath.append(optionService.getBlogBaseUrl());
        }

        fullPath.append(URL_SEPARATOR);

        if (permalinkType.equals(PostPermalinkType.DEFAULT)) {
            fullPath.append(archivesPrefix)
                .append(URL_SEPARATOR)
                .append(post.getSlug())
                .append(pathSuffix);
        } else if (permalinkType.equals(PostPermalinkType.ID)) {
            fullPath.append("?p=")
                .append(post.getId());
        } else if (permalinkType.equals(PostPermalinkType.DATE)) {
            fullPath.append(DateUtils.year(post.getCreateTime()))
                .append(URL_SEPARATOR)
                .append(monthString)
                .append(URL_SEPARATOR)
                .append(post.getSlug())
                .append(pathSuffix);
        } else if (permalinkType.equals(PostPermalinkType.DAY)) {
            fullPath.append(DateUtils.year(post.getCreateTime()))
                .append(URL_SEPARATOR)
                .append(monthString)
                .append(URL_SEPARATOR)
                .append(dayString)
                .append(URL_SEPARATOR)
                .append(post.getSlug())
                .append(pathSuffix);
        } else if (permalinkType.equals(PostPermalinkType.YEAR)) {
            fullPath.append(DateUtils.year(post.getCreateTime()))
                .append(URL_SEPARATOR)
                .append(post.getSlug())
                .append(pathSuffix);
        } else if (permalinkType.equals(PostPermalinkType.ID_SLUG)) {
            fullPath.append(archivesPrefix)
                .append(URL_SEPARATOR)
                .append(post.getId())
                .append(pathSuffix);
        }
        return fullPath.toString();
    }
}
```