Bonus - Public & Private Bookshelves:

PublicBook Shelves Entity Description

Overview

Implemented functionality within the app to enable users to mark their bookshelves as either private or public.

- Defined "private" bookshelves as accessible only to the respective user who created them, ensuring confidentiality and privacy of personal reading collections.
- Designated "public" bookshelves as accessible to all users of the common library system, enabling others to view the books within these shelves in a read-only mode.

Consider `Tag.java`, a new field called `isPublic` has been added. Now users can set a tag to be public or private. All public tags will be displayed on the page "Public Books and Bookshelves" in the navigation bar. When a particular tag is selected on that page, only those books associated with that tag get filtered and displayed. This is a global page where all tags are displayed.

Additionally, the books are in read-only mode, so inner information cannot be accessed.

Details of Implementation and Changes to Existing Code

`Tag.java` (com.sismics.books.core.model.jpa)

```
@Entity
@Table(name = "T_TAG")
public class Tag {
    /**
     * Is Public or Private.
     */
    @Column(name = "TAG_PUBLIC_C", nullable = false, length = 36)
    private boolean isPublic;

    // getter and setter of the new attribute
    public boolean getisPublic() {
        return isPublic;
    }

    public void setisPublic(boolean aPublic) {
        isPublic = aPublic;
    }
}
```

`BookListingResource.java` (com.sismics.books.rest.resource)

```
    @GET
    @Path("tags")
    @Produces(MediaType.APPLICATION_JSON)
    public Response listbytags(
            @QueryParam("limit") Integer limit,
            @QueryParam("offset") Integer offset,
            @QueryParam("sort_column") Integer sortColumn,
            @QueryParam("asc") Boolean asc,
            @QueryParam("search") String search,
            @QueryParam("read") Boolean read,
            @QueryParam("tagId") String tagId) throws JSONException {

        if (!authenticate()) {
            throw new ForbiddenClientException();
        }

        JSONObject response = new JSONObject();
        List<JSONObject> books = new ArrayList<>();

        UserBookDao userBookDao = new UserBookDao();
        TagDao tagDao = new TagDao();
        PaginatedList<UserBookDto> paginatedList =
PaginatedLists.create(limit, offset);
        SortCriteria sortCriteria = new SortCriteria(sortColumn, asc);
        UserBookCriteria criteria = new UserBookCriteria();
        criteria.setSearch(search);
        criteria.setRead(read);
        if (tagId != null) {
            Tag tag = tagDao.getById(tagId);
            if (tag != null) {
                criteria.setUserId(tag.getUserId());
                criteria.setTagIdList(Lists.newArrayList(tag.getId()));
            }
        }
        try {
            userBookDao.findByCriteria(paginatedList, criteria,
sortCriteria);
        } catch (Exception e) {
            throw new ServerException("SearchError", "Error searching in
books", e);
        }
        for (UserBookDto userBookDto : paginatedList.getResultList()) {
            JSONObject book = new JSONObject();
            book.put("id", userBookDto.getId());
            book.put("title", userBookDto.getTitle());
            book.put("subtitle", userBookDto.getSubtitle());
            book.put("author", userBookDto.getAuthor());
            book.put("language", userBookDto.getLanguage());
            book.put("publish_date", userBookDto.getPublishTimestamp());
            book.put("create_date", userBookDto.getCreateTimestamp());
            book.put("read_date", userBookDto.getReadTimestamp());
            List<TagDto> tagDtoList =
tagDao.getByUserBookId(userBookDto.getId());
```

```
            List<JSONObject> tags = new ArrayList<>();
            for (TagDto tagDto : tagDtoList) {
                JSONObject tag = new JSONObject();
                tag.put("id", tagDto.getId());
                tag.put("name", tagDto.getName());
                tag.put("color", tagDto.getColor());
                tag.put("isPublic",tagDto.getisPublic());
                tags.add(tag);
            }
            book.put("tags", tags);

            books.add(book);
        }
        response.put("total", paginatedList.getResultCount());
        response.put("books", books);

        return Response.ok().entity(response).build();
    }
```

`TagResource.java` (com.sismics.books.rest.resource)

```
    @GET
    @Path("/list-public")
    @Produces(MediaType.APPLICATION_JSON)
    public Response listPublic() throws JSONException {
        if (!authenticate()) {
            throw new ForbiddenClientException();
        }
        TagDao tagDao = new TagDao();
        UserDao userDao = new UserDao();
        List<Tag> tagList = tagDao.getPublicTags();
        JSONObject response = new JSONObject();
        List<JSONObject> items = new ArrayList<>();
        for (Tag tag : tagList) {
            User user = userDao.getById(tag.getUserId());
            System.out.println(user.getUsername());

            JSONObject item = new JSONObject();
            item.put("id", tag.getId());
            item.put("name", tag.getName());
            item.put("color", tag.getColor());
            item.put("isPublic",tag.getisPublic());
            item.put("user", user.getUsername());
            items.add(item);
        }
        response.put("tags", items);
        return Response.ok().entity(response).build();
    }
```