**User Management Documentation**

As we understand, we have to fix the roles of Reader, Architect, Repository Admin and Super Admin (Here is a confusion that it will be fixed or Super Admin will set the roles of Reader, Architect and Repository Admin of each company separately.)

**Reader:**

* Objects
  + Can only read forms of Elements, Relationships and Connectors but can’t Create, Update or Delete.
* Models
  + Can only open and see Canvas, Catalogs and Matrix from tree which are already created but can’t modify or create new.
* User Profile
  + Can only view his own profile and modify his own profile.

**Architect:**

* Objects
  + Can Create, Read, Update and Delete forms of Elements, Relationships and Connectors.
* Models
  + Can Create, Read, Update and Delete Canvas, Catalogs and Matrix from tree.
* Import/Export
  + Can Import and Export Database.
* User Profile
  + Can only view his own profile and modify his own profile.

**Repository Admin:**

* Objects
  + Can Create, Read, Update and Delete forms of Elements, Relationships and Connectors.
* Models
  + Can Create, Read, Update and Delete Canvas, Catalogs and Matrix from tree.
* Import/Export
  + Can Import and Export Database.
* Admin
  + Can Create, Read, Update and Delete users (Architect and Reader).
  + Can create a backup and restore.
* User Profile
  + Can view his own profile and modify his own profile.

**Super Admin:**

* Objects
  + Can Create, Read, Update and Delete forms of Elements, Relationships and Connectors.
* Models
  + Can Create, Read, Update and Delete Canvas, Catalogs and Matrix from tree.
* Import/Export
  + Can Import and Export Database.
* Admin
  + Can Create, Read, Update and Delete users (Repository Admin, Architect and Reader).
  + Can create a backup and restore.
* User Management
  + Can Create, Read, Update and Delete User Roles.
* User Profile
  + Can view his own or any user profile and modify his own profile and any user profile.

**Real World Practices:**

**General Details of User Management System**:

User management involves defining and managing users, roles, and their access levels in a system. A user management dashboard or console provides system administrators with a high-level view of a system's active user sessions, their login statuses, the privileges of each user, and their activity in the system. It enables system admins to make business-critical, real-time security decisions. A typical user management implementation involves a wide range of functionality such as adding/deleting users, controlling user activity through permissions, managing user roles, defining authentication policies, managing external user stores and manual/automatic logout, and resetting passwords.

Reference: <https://docs.wso2.com/display/OB150/Introduction+to+User+Management>

**LMS (Learning Management System):**

We have LMS in almost all universities which have different types of users as follows:

1. **LMS Administrator role:** An administrator in LMS (admin) is a user role with the highest number of privileges. They have access to the admin dashboard. That’s where you can find all settings and features. Admins can modify the LMS configurations, create users, and manage content in the LMS.
2. **Manager role:** A manager role in an LMS application will be limited compared to the administrator role capabilities. For example, one manager may be entitled to oversee all eLearning content related to Science subject groups. Another manager may be entitled to oversee online learning content over language subjects – English, French, Spanish, and the like.
3. **Teacher role (Instructor):** The ‘teacher’ role in an LMS is oriented to teaching. You as a teacher can create course content. Courses can be managed and you can receive assignments from students online. You can schedule instructor-led classes where an instructor will be present for assistance in the eLearning session. As part of course management, you as a teacher will be able to monitor courses, manage reviews, answer student questions, enrollments, and run relevant reports for insights. You will be able to mark courses as ‘completed’ for students who have completed the courses. Some of the LMS’s enable limited access for external trainers.
4. **Learner role:** We’ve all been learners at some point in time – at school, college, or even corporate training at work. This role will have the highest number of users in the application usually. The learner role will have its dashboard that will show access to courses enrolled. It also shows your progress in the courses. You as a learner just have to focus on completing your courses and learning paths.

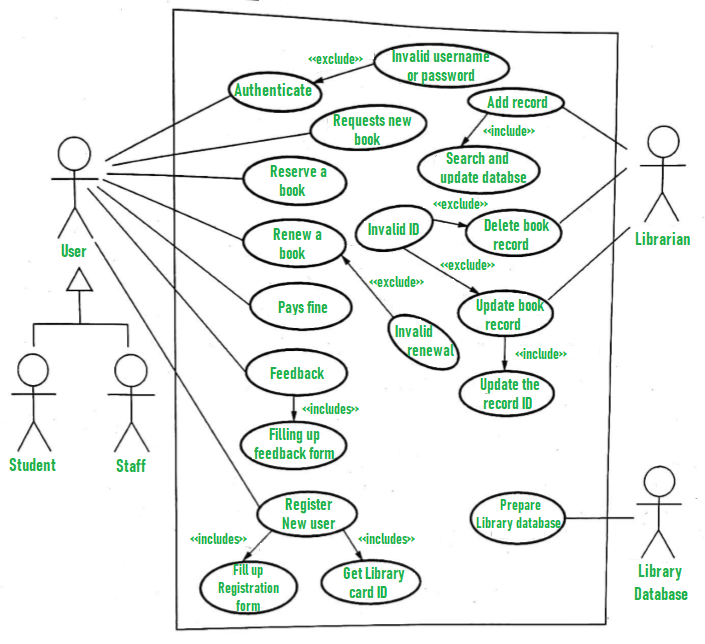
For detailed reference: <https://edly.io/blog/how-does-a-learning-management-system-work/>

**CRM (Customer Relationship Management):**

This is a ZOHO CRM pdf file which has defined user management in detail with examples and pictorial representation.

<https://www.zoho.com/sites/default/files/crm/user-management.pdf>

**LIBRARY MANAGEMENT SYSTEM UML DIAGRAM**



**DETAILS OF OUR PROJECT**

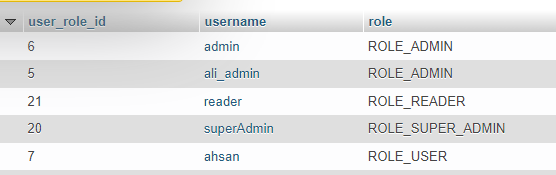
**Database structure:**

**User Table:**



**User Roles:**

* **ROLE\_ADMIN: Repository Admin**
* **ROLE\_READER: Reader**
* **ROLE\_USER: Architect**
* **ROLE\_SUPER\_ADMIN: Super Admin (EAXEE Admin)**

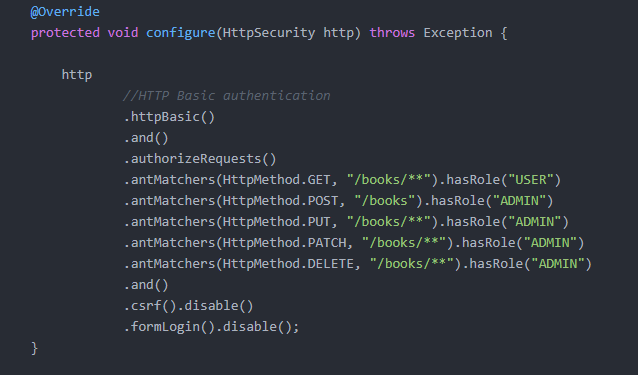


**SPRING SECURITY**

Spring framework provides a library named Spring Security which has user management with different roles having different authorities.

**Reference:**

* <https://mkyong.com/spring-boot/spring-rest-spring-security-example/>



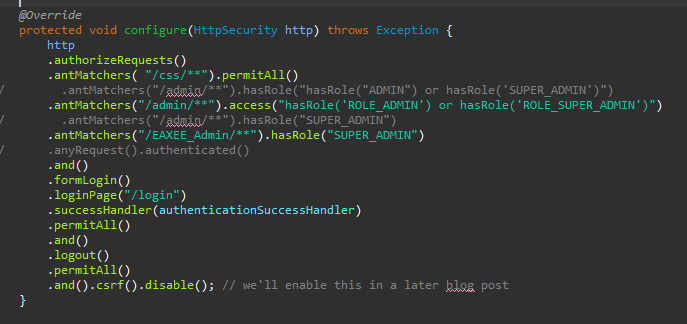
**More references:**

[**https://spring.io/guides/gs/securing-web/**](https://spring.io/guides/gs/securing-web/)

<https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html>

<https://www.sciencedirect.com/topics/computer-science/user-management>

**Our Configuration Code (What we have already done in EAXee):**



**For backend we can define urls for different type of user:**

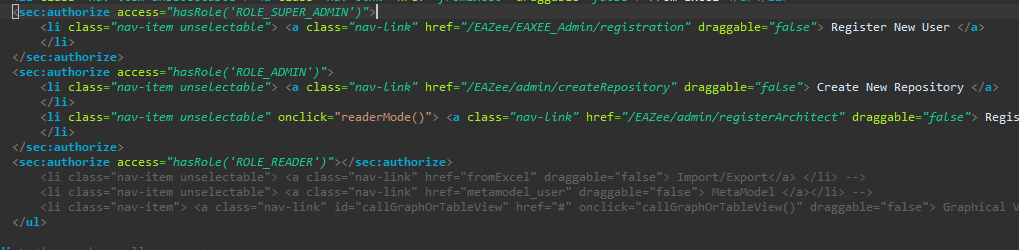
.antMatchers("/admin/\*\*").access("hasRole('ROLE\_ADMIN') or hasRole('ROLE\_SUPER\_ADMIN')")

Any url starting from this [https://eaxee.com/EAZee/admin/\*\*](https://eaxee.com/EAZee/admin/**) can only be accessed by user having role ”ROLE\_ADMIN” or “ROLE\_SUPER\_ADMIN”.

.antMatchers("/EAXEE\_Admin/\*\*").hasRole("SUPER\_ADMIN")

Any url starting from this [https://eaxee.com/EAZee/EAXEE\_Admin/\*\*](https://eaxee.com/EAZee/EAXEE_Admin/**) can only be accessed by user having role “ROLE\_SUPER\_ADMIN”.

**For Front End:**



**For front end we can define section for different users:**

<**sec:authorize** access=*"hasRole('ROLE\_ADMIN')"*>

Any content inside this section like forms, tables, buttons, canvas and other front-tags will show only to role: ROLE\_ADMIN.

<**sec:authorize** access=*"hasRole('ROLE\_READER')"*></**sec:authorize**>

Any content inside this section like forms, tables, buttons, canvas and other front-tags will show only to role: ROLE\_READER.

<**sec:authorize** access=*"hasRole('ROLE\_SUPER\_ADMIN')"*>

Any content inside this section like forms, tables, buttons, canvas and other front-tags will show only to role: ROLE\_SUPER\_ADMIN.

**Summary:**

1. There are no changes in the database if we will working on fixed user roles in our application.
2. We are working on the same way through which we are seeing on the above reference links.
3. For frontend we are using same page with multiple sections for different user roles.
4. For Backend we are using user roles to access to different pages according to the user role.