

Library Management System

A CS814 Course Project Report

Submitted by

Kushagra Gupta (202IS015)
Surya Prakash Gupta (202IS026)

Department of Computer Science and Engineering
National Institute of Technology Karnataka
P.O. Srinivasnagar, Surathkal, Mangalore-575025
Karnataka, India
January 2021

Introduction	3
Types of Users	3
The Functionality of Users	3
Authorization	6
The need for RBAC based authorization	6
Components of RBAC	8
Users and roles	8
Permissions	8
Sessions	9
Components of the administrative model	11
Conclusion	11
References	11

Introduction

In today's time, everything is getting digital, fast, modernized, and to stay competitive with everyone around us, we require easy and correct access to things. The library management system is an application that helps students, faculties to manage their books related information precisely in one place.

In our library management system, we have a *home page* from where a user can enter. Next to that, we have created a *login page* that is the same for all the users regardless of their roles. After the user login, the authentication system will detect his/her role. After the login page, we have a dashboard of the user. In the dashboard, all the permissions are specified in terms of functionalities that are authorized to that particular user. Each user can access his/her permission/function according to the assigned role.

Types of User Roles

Our library management system consists of three different types of user roles and each one is having different functionalities.

These are the different roles which we have included in our management system

1. Student
2. Faculty
3. Staff

The Functionality of the Users Roles

Let's first discuss the role or functionality of the *student* here. As normally what is possible for a student in a library management system? They can view, issue, and return the books. So keeping this in mind we have provided all such functionalities to this particular set of users. Despite this, we have also provided an option to a student that he/she can also view those books that he/she has issued. There is a restriction that a particular student cannot view the list of other students.

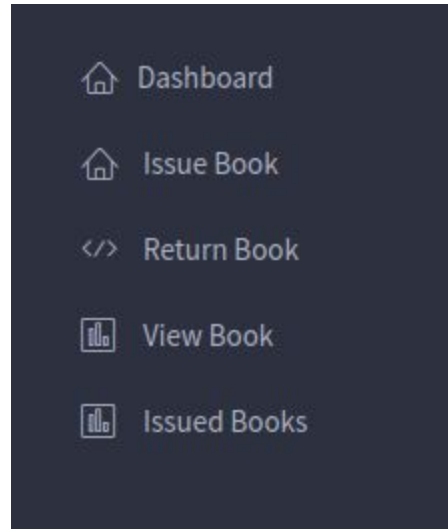


Fig. *Permissions* assigned to users having a role '*Student*'

Faculty is another role in this management system in addition to the *student*. We have provided them with the view and issue of the books. They can also view issued books as well as make a return of the book. All these options are similar to a student user. What extra power do they have? They have given extra privilege apart from the students' functionalities. They can view all the students apart from themselves. This is what additional functionality that we have implemented in the management system for the faculty members.

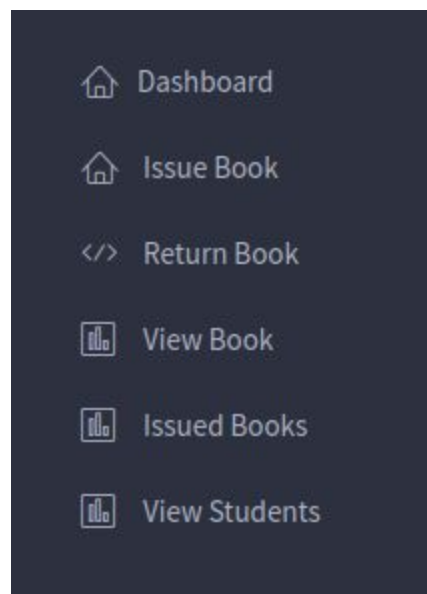


Fig. *Permissions* assigned to users having role '*Faculty*'

Now comes the third role of user i.e *staff*. What permissions can a staff member have? We have tried to inherit almost all such functionalities under this type of user. Specifically, we have granted the following permissions to the role of *Staff*:

- View Book
- Add Book
- Delete Book
- Add Student
- View Student
- View Faculty

Similar to the faculty user, the librarian can also view all the students, and in addition to that he/she is given the power to view all the faculty members as well. There is one special functionality that we have added under the librarian i.e he/she can add students as well.

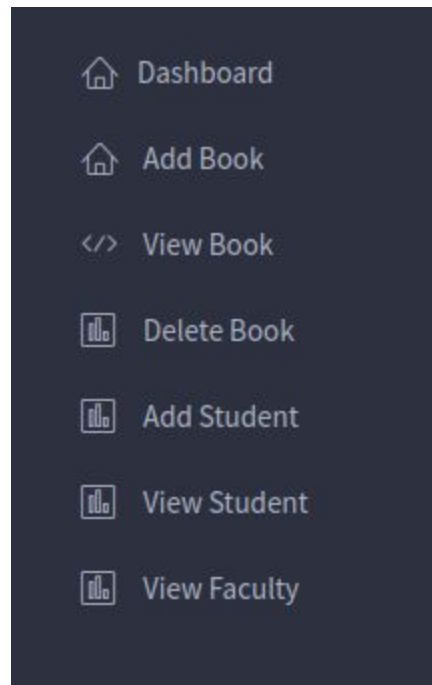


Fig. *Permissions* assigned to users having role '*Staff*'

Authorization

What is authorization? When we specify access rights or privileges, which is related to general information security and to *access control* in particular. The system uses the access control rules to decide whether access requests from (authenticated) users shall be approved (granted) or disapproved (rejected).

Access control relies on access policies. The access control process can be divided into the following phases

- ❖ Policy definition phase where access is authorized
- ❖ Policy enforcement phase where access requests are approved or disapproved.

Authorization is the function of the policy definition phase which precedes the policy enforcement phase where access requests are approved or disapproved based on the previously defined authorizations.

The need for RBAC based authorization

Role-based access control (RBAC) is an approach to restricting system access to authorized users. Many applications include *role-based access control* (RBAC) and thereby rely on authorization. Access control also uses *authentication* to verify the identity of *different users*. When a user tries to access a resource, his/her identity will be checked. If the user is authenticated, then he/she will be given a successful right to enter into the system otherwise not.

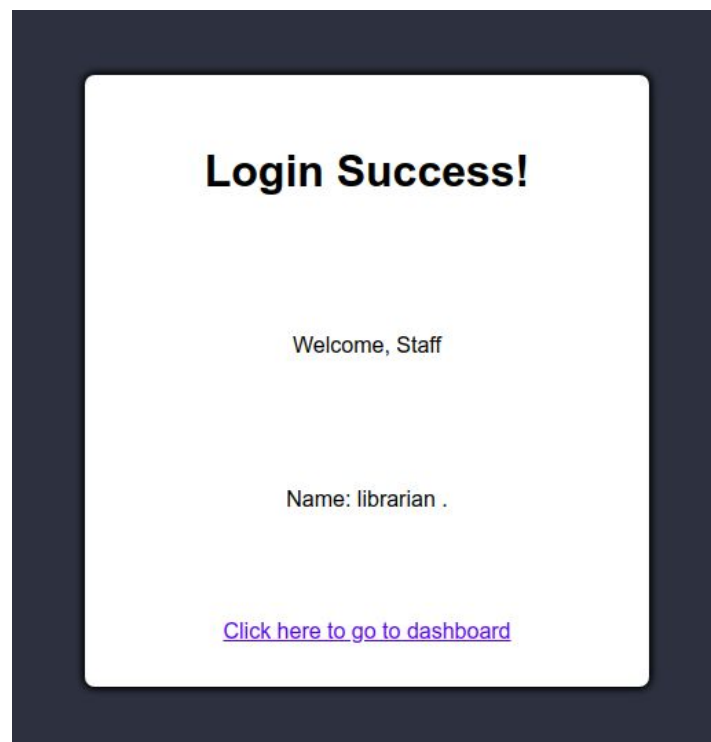


Fig. *User authentication successful*

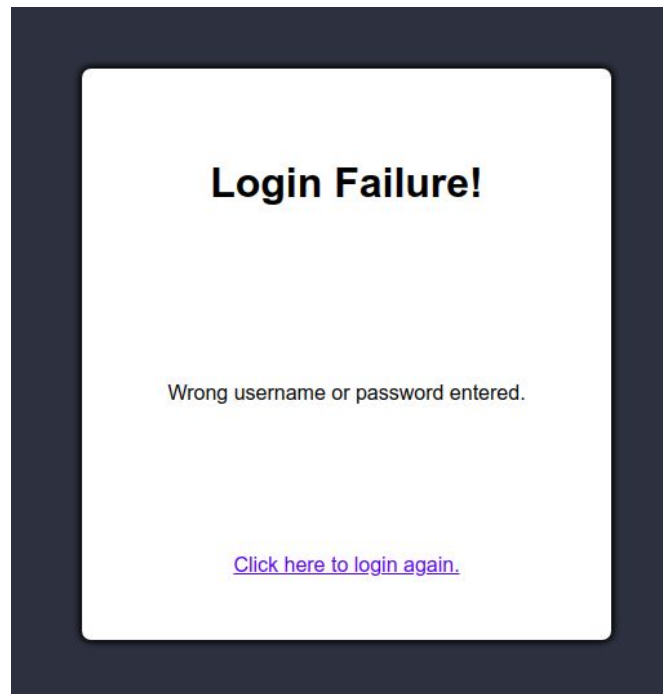


Fig. *User authentication failed.*

When a user tries to access a resource, the access control process checks that the user has been authorized to use that resource. RBAC includes the capability to establish relations between roles, between permissions and roles, and between users and roles. With RBAC, role-permission relationships can be pre-defined, which makes it simple to assign users to predefined roles. Without RBAC, it can also be difficult to determine what permissions have been authorized for what users. An access control policy is included in RBAC components such as role-permission, user-role, and role-role relationships. These components collectively determine whether a particular user is allowed access to a certain piece of system data. RBAC components can be configured directly by the system administrator or indirectly by appropriate roles as delegated by the system administrator. The policy enforced in a given system results from the specific configuration of RBAC components as directed by the system administrator. Because the access control policy can, and usually does, change over the system life cycle, RBAC offers a key benefit through its ability to modify access control to meet changing application needs.

```

{% if user.role == 'Student' %}
    <li><a href="/issuebook" class=""><i class="lnr lnr-home"></i> <span>Issue Book</span>
    </a></li>
    <li><a href="/returnbook" class=""><i class="lnr lnr-code"></i> <span>Return Book</span>
    </a></li>
    <li><a href="/viewbook" class=""><i class="lnr lnr-chart-bars"></i> <span>View Book</
    span></a></li>
    <li><a href="/issuedbook" class=""><i class="lnr lnr-chart-bars"></i> <span>Issued
    Books</span></a></li>
{% elif user.role == 'Faculty' %}
    <li><a href="/issuebook" class=""><i class="lnr lnr-home"></i> <span>Issue Book</span>
    </a></li>
    <li><a href="/returnbook" class=""><i class="lnr lnr-code"></i> <span>Return Book</span>
    </a></li>
    <li><a href="/viewbook" class=""><i class="lnr lnr-chart-bars"></i> <span>View Book</
    span></a></li>
    <li><a href="/issuedbook" class=""><i class="lnr lnr-chart-bars"></i> <span>Issued
    Books</span></a></li>
    <li><a href="/viewstudent" class=""><i class="lnr lnr-chart-bars"></i> <span>View
    Students</span></a></li>
{% else %}
    <li><a href="/addbook" class=""><i class="lnr lnr-home"></i> <span>Add Book</span></a>
    </li>
    <li><a href="/viewbook" class=""><i class="lnr lnr-code"></i> <span>View Book</span></a>
    </li>
    <li><a href="/deletebook" class=""><i class="lnr lnr-chart-bars"></i> <span>Delete Book
    </span></a></li>
    <li><a href="/addstudent" class=""><i class="lnr lnr-chart-bars"></i> <span>Add Student
    </span></a></li>
    <li><a href="/viewstudent" class=""><i class="lnr lnr-chart-bars"></i> <span>View
    Student</span></a></li>
    <li><a href="/viewfaculty" class=""><i class="lnr lnr-chart-bars"></i> <span>View
    Faculty</span></a></li>
{% endif %}

```

Fig. Here, we are authorizing the permissions respective to the roles of the user. The hyperlinks denoting the permission are accessible only if the user is authorized to access that permission.

Components of RBAC

Role-based access control has four models as shown below but we have implemented $RBAC_0$ in our application. $RBAC_0$ consists of users (U), roles (R), permissions (P), and sessions.

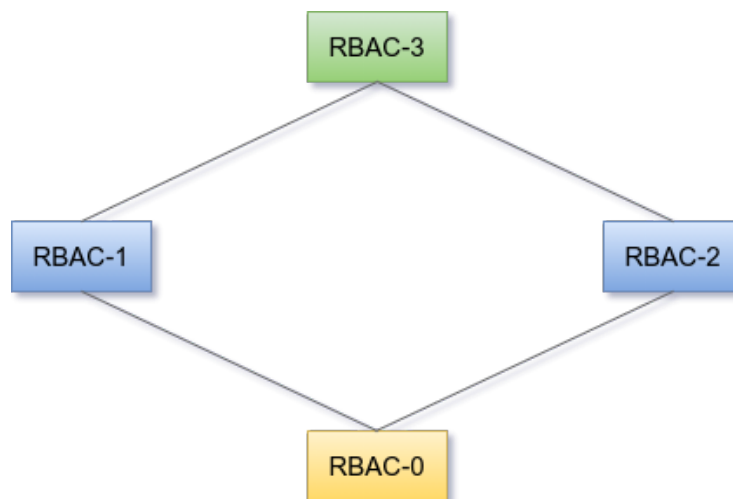


Fig. A family of RBAC models

Users and roles

In our application, a user is a member of the college administration having a role assigned from a defined set of roles which are *student*, *faculty*, or *staff*. A role is a named job function within the organization that describes the authority and responsibility conferred on a member of the role. In our application's database, we have created a table of users. Each user is assigned a role using the role attribute present in the user table so as to facilitate the *user-role assignment*.

Permissions

A permission is an approval of a particular mode of access to one or more objects in the system. Permission is also known as authorization, access right, and privilege. Permissions are always positive and confer on their holder the ability to act as the system.

In our application, we have defined each permission in terms of a function which a user will perform. The various permissions present in our system are:

- Add Book
- View Book
- Add Student
- View Student
- View Faculty
- Return Book
- Issue Book
- Delete Book

Admin User: An admin is a type of user who has access to all the functionalities present in the management system. Apart from the above permissions, the administrator or admin has the following permissions:

- Add a new user
- Assign a role to a user
- Modify the role of any user
- Remove user
- Modify/Delete user

Sessions

Users establish sessions during which they may activate a subset of the roles they belong to. Each session maps one user to possibly many roles. The permissions available to the user are the union of permissions from all roles activated in that session. Each session is associated with a single user. A user might have multiple sessions open simultaneously, each in a different browser. In the RBAC model, the user's discretion alone determines which roles are activated in a given session. This model also lets roles be dynamically activated and deactivated during a

session. Sessions are under the control of individual users. As far as the model is concerned, a user can create a session and choose to activate some subset of the user's roles.

In our application, we have created a session token *username* when the user login to the system. As soon as the user is authenticated, the session token is transmitted to all the functions using which the user can access the permission assigned to him/her.

```
def login(request):  
    if request.method == 'POST':  
        form = LoginForm(request.POST)  
        if form.is_valid():  
            username = form.cleaned_data.get('username')  
            password = form.cleaned_data.get('password')  
            request.session['username'] = username  ← Creating username session  
            try:  
                user = Users.objects.get(username = username)  
            except:  
                user = None  
            if not user:  
                isValid = False  
                return render(request, 'loginfailure.html', {'isValid':isValid})  
            if user.password == password:  
                isValid = True  
                return render(request, 'login_success.html', {'user':user, 'isValid':isValid})  
            else:  
                isValid = False  
                return render(request, 'loginfailure.html', {'isValid':isValid})  
        else:  
            return HttpResponse("Invalid form")  
    else:  
        form = LoginForm()  
    return render(request, 'login.html', {'form':form})
```

Fig. Creating a session token 'username' as soon as the user login into the system. The same session token will be transmitted to all the functions

```
def returnbook(request):
    username = request.session.get('username')
    user = Users.objects.get(username=username)

    #If no books issued
    ub = UserBook.objects.all()
    if not ub:
        return render(request, 'returnbook.html', {'user':user, 'ub':False})

    ub = UserBook.objects.filter(username=username)
    if not ub:
        return render(request, 'returnbook.html', {'user':user, 'ub':False})

    isbnList = []
    for i in ub:
        isbnList.append(i.isbn)
    book = Book.objects.filter(isbn__in = isbnList)
```

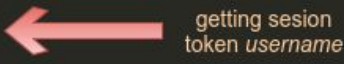


Fig. Getting the session token *username* in the *returnbook* function

```
def deletebook(request):
    username = request.session.get('username')
    book = Book.objects.all()
    user = Users.objects.get(username=username)

    if request.method == 'POST':
        isbn = request.POST.get('delete')
        Book.objects.get(isbn = isbn).delete()

        #removing all the books issued with the above isbn
        UserBook.objects.filter(isbn = isbn).delete()

    return render(request, 'deletebook.html', {'book':book, 'user':user})
```




Fig. Getting the session token *username* in the *deletebook* function

Components of the administrative model

Administration of RBAC is very important and must be carefully controlled to ensure that policy does not drift away from its original objectives. Managing these roles and users, and their interrelationships is a dangerous task. Administration of RBAC involves control over different components, including creation and deletion of roles, creation, and deletion of permissions, assignment of permissions to roles and their removal, creation, and deletion of users, assignment of users to roles, and their removal.

The screenshot displays an administrative interface for user management. On the left, a sidebar contains two main sections: 'AUTHENTICATION AND AUTHORIZATION' with links for 'Groups' and 'Users' (each with a '+ Add' button), and 'LIBRARY' with links for 'Books', 'User books', and 'Userss' (each with a '+ Add' button). The 'Userss' link is highlighted in yellow. The main content area is titled 'Change users' and features a 'HISTORY' button in the top right. The form contains the following fields: 'Firstname' (surya), 'Lastname' (gupta), 'Username' (surya123), 'Password' (student@21), 'Email' (surya.2021s026@nitk.edu.in), 'Branch' (CSE), and 'Role' (a dropdown menu currently showing 'Student' with a list of options: Student, Staff, Faculty, and Student). At the bottom of the form, there are four buttons: a red 'Delete' button, and three blue buttons labeled 'Save and add another', 'Save and continue editing', and 'SAVE'.

Fig. This is the administrative panel. Here the administrator(admin) can assign or revoke roles from the user and perform many other functions stated above.

Conclusion

The role-based access control model has three components: users, roles, and permissions. When a user is assigned a role, we have the user-role relationship and when a role is assigned a permission we have a role-permission relationship. In our application, we have created a set of users and assigned them respective roles. Each user is authorised to access a permission according to their role. This fulfills the requirement of RBAC components. We have a user with an admin role who has access to all of the permissions present in the system. This admin role fulfills the requirement of the administrative component of RBAC. Using RBAC as an access control model, we were able to successfully implement all of the components required in our application.

References

1. Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.* 2, 1 (Feb. 1999), 105–135. DOI:<https://doi.org/10.1145/300830.300839>
2. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (February 1996), 38–47. DOI:<https://doi.org/10.1109/2.485845>
3. <https://docs.djangoproject.com/en/3.1/contents/>