User Login and Authorization in Rate My Course

Gary Wit; Rate My Course Team

Abstract—This paper explains how the application implements user login and authorization using Django sessions and cookies. It details browser cookie behavior, server-side session and token issuance, session storage and association with users, and password verification.

Index Terms—Authentication, Authorization, Sessions, Cookies, CSRF, Django.

I. Browser Cookies and AJAX Request Authorization

The browser stores two key cookies: sessionid (session identifier) and csrftoken (CSRF protection token). All subsequent requests automatically include sessionid. For state-changing requests, the app embeds csrftoken in forms and sets X-CSRFToken on AJAX calls.

Login and forms include CSRF tokens, and protected AJAX endpoints read the CSRF header. Below is a real form and fetch example from the app.

```
{% extends "base.html" %}

{% block title %}■■ - ■■■■■■{% endblock %}

{% block content %}
<div class="container">
    <div class="auth-container">
        <div class="auth-card">
            <h2><i class="fas fa-sign-in-alt"></i> ■■</h2>
            <form method="POST" action="{% url 'login' %}">
                {% csrf_token %}
                <div class="form-group">
                    <label for="username">■■■</label>
                    <input type="text" id="username" name="username" required>
                </div>
                <div class="form-group">
                    <label for="password">■■</label>
                    <input type="password" id="password" name="password" required>
                </div>
                <button type="submit" class="btn btn-primary">■■</button>
            </form>
            <p class="auth-link">■■■■■■ <a href="{% url 'register' %}">■■■■</a></p>
        </div>
    </div>
</div>
{% endblock %}
            {% csrf_token %}
            <input type="hidden" name="entity_type" id="report-entity-type">
            <input type="hidden" name="entity_id" id="report-entity-id">
            <div class="form-group">
                <label for="reason">■■■■</label>
                <textarea id="reason" name="reason" rows="4" required></textarea>
            </div>
            <button type="submit" class="btn btn-primary">■■■■</button>
        </form>
    </div>
</div>
{% endblock %}
```

```
{% block scripts %}
<script>
document.addEventListener('DOMContentLoaded', function() {
    // Favorite toggle
    const favoriteBtn = document.getElementById('favorite-btn');
    if (favoriteBtn) {
        favoriteBtn.addEventListener('click', function() {
            const courseId = this.dataset.courseId;
            fetch('{% url "toggle_favorite" course_id=course.course_id %}', {
                method: 'POST',
                headers: {'X-CSRFToken': '{{ csrf_token }}'}
            })
            .then(response => response.json())
            .then(data => {
                if (data.action === 'added') {
                    this.classList.add('btn-favorite');
                    this.innerHTML = '<i class="fas fa-heart"></i> ■■■';
                } else {
                    this.classList.remove('btn-favorite');
                    this.innerHTML = '<i class="fas fa-heart"></i> ■■';
                }
            });
        });
    }
```

II. Server Identification and Cookie/Token Issuance

Upon successful authentication, the server calls login(request, user). The session middleware persists a server-side session and returns Set-Cookie: sessionid to the browser. Authentication middleware maps session data to request.user on future requests.

```
        messages.error(request, "■■■■■")
        return redirect("courses")

    existing = Rating.objects.filter(user_id=request.user.id, course_id=course_id).first()
    if existing:
        messages.info(request, "■■■■■■■■■■■■■■■")
        return redirect("course_detail", course_id=course_id)

    r = Rating(
        user_id=request.user.id,
        course_id=course_id,
```

III. Session Storage and Association with Users

Django stores active sessions in the django_session table. Each session record links to a session key (sessionid) and serialized session data (including the authenticated user ID). AuthenticationMiddleware reconstructs request.user from the session on each request.

Access control uses user.is_authenticated for general checks and user.is_staff for administrative views. The app also uses @login_required to guard write operations such as rating, comments, reactions, favorites, and reports.

```
        comment_text=request.POST.get("comment_text", ""),
        anonymous_flag=request.POST.get("anonymous_flag") == "on",
        created_at=timezone.now(),
```

```
    )
    r.save()

    tag_names = request.POST.getlist("tags")
    for t in tag_names:
        name = (t or "").strip()
        if not name:
            continue
        tag_obj, _ = Tag.objects.get_or_create(name=name)
        CourseTag.objects.create(course_id=course_id, tag_id=tag_obj.tag_id, user_id=request.user.id,

    messages.success(request, "████████")
    return redirect("course_detail", course_id=course_id)

@login_required
def add_comment(request: HttpRequest, rating_id: int):
```

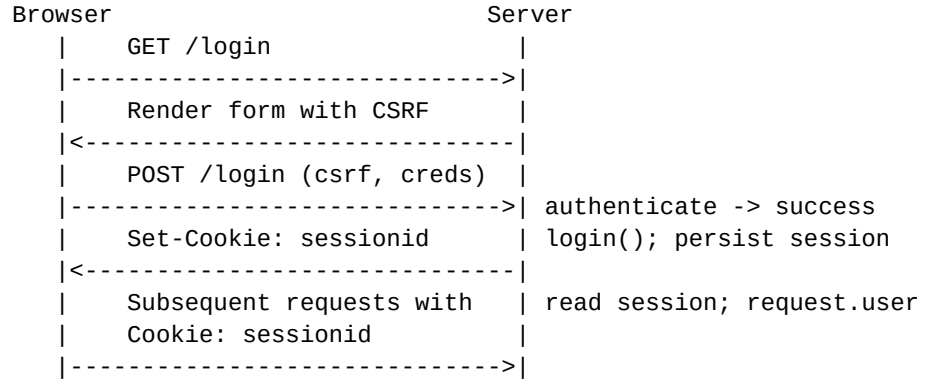
IV. Password Verification

Registration stores user credentials in the auth_user table using the framework's password hashing (PBKDF2 by default). During login, authenticate() verifies the supplied password by hashing and comparing against the stored hash.

```
def login_view(request: HttpRequest):
    if request.method == "POST":
        username = request.POST.get("username", "")
        password = request.POST.get("password", "")
        user = authenticate(request, username=username, password=password)
        if user:
            login(request, user)
            next_url = request.GET.get("next")
            return redirect(next_url or "index")
        messages.error(request, "█████████")
    return render(request, "login.html")

def logout_view(request: HttpRequest):
    logout(request)
    return redirect("index")
@login_required
def rate_course(request: HttpRequest, course_id: int):
    try:
```

V. Authentication Flow Diagram



```
|    POST /favorite (AJAX)      | CSRF header + session
|------------------------------>|
|    GET /logout                | logout(); expire session
|------------------------------>| Set-Cookie expires
|<------------------------------|
```

VI. Security Considerations

CSRF protection is enforced for all state-changing requests. Session cookies are HttpOnly and SameSite=Lax by default in development. For production, Secure and stricter SameSite can be enabled to reduce risk. No plaintext passwords are stored; password hashing and salting are handled by the framework.

References

[1] Django Documentation: Authentication, Sessions, CSRF