Phase 2 Report: User Management and Security Implementation

Date: October 8, 2024 - October 29, 2024 Group 18

I.Summary of the Work Done to Date

1.1 User Authentication and Registration

We successfully implemented Django's built-in authentication system, which manages user registration, login, and session handling. This system securely hashes user credentials and offers session management, ensuring smooth browsing for logged-in users.

Features implemented:

Registration: We used Django's UserCreationForm to create a secure registration process. New users can register with a valid email address.

```
class LoginForm(forms.Form):  2 用法
    email=forms.EmailField(error_messages={'required':'请输入邮箱','invalid':'请输入有效的邮箱'})
    password = forms.CharField(min_length=6,max_length=20,error_messages={
        'required':'请输入密码',
        'min_length':'密码的最小长度为6个字符',
        'max_length':'密码的最大长度为20个字符'
    })
```

Login & Logout: The login and logout functionality was implemented using Django's authentication views, which restrict access to certain pages based on the user's authentication status.

Session Management: Django's session middleware ensures user session data is securely stored, allowing for smooth page transitions after login.

1.2 Login and Registration Interface Design

We designed and implemented the login and registration interfaces using Django's Template Language (DTL). These interfaces ensure a user-friendly experience while tightly integrating with the backend authentication logic.

```
<div style="..." class="m-auto">
    <h1>请登录</h1>
    <form action="" method="post">
        <div class="md-3">
            <label class="mb-1 mt-1">邮箱</label>
            <input type="email" name="email" class="form-control" placeholder="请输入邮箱">
        </div>
        <div class="md-3">
            <label class="mb-1 mt-1">密码</label>
            <input type="password" name="password" class="form-control" placeholder="请输入密码">
        </div>
    </form>
    <div class="form-check mt-2">
        <input class="form-check-input" type="checkbox" name="remember " value="1" id="flexCheckDefault">
        <label class="form-check-label" for="flexCheckDefault">
            记住我
        </label>
    </div>
```

Login Interface: Users can log in using their email and password, and errors are displayed

promptly for any incorrect inputs.

Registration Interface: The registration form is integrated with CAPTCHA to enhance security, ensuring that only legitimate users can create accounts.

1.3 Security Measures

Security is a core focus of this phase. To prevent unauthorized access and enhance the platform's security, the following measures were implemented:

CAPTCHA Integration: To prevent spam and automated submissions, we integrated CAPTCHA into the registration process. Users must complete the CAPTCHA challenge before registering.

Email Verification: After registration, users must verify their email addresses. We implemented an email verification system where users receive an activation link via email, and accounts are activated only after clicking the link. This ensures that only valid users can access the platform.

CSRF Protection: We enabled Django's built-in CSRF protection across forms to prevent cross-site request forgery attacks.

1.4 Email Sending Functionality

To implement the verification email sending feature during user registration, we integrated QQ Mail's SMTP service into the system. The process is as follows:

Email Configuration: In QQ Mail settings, we enabled POP3/IMAP/SMTP services and obtained the authorization code for third-party applications.

SMTP Configuration: In Django's settings.py, we configured the SMTP service for secure email sending using QQ Mail.

The mail server is configured with smtp.qq.com, and authentication is performed using the authorization code, ensuring secure and stable email transmission.

With this setup, we successfully implemented the email verification functionality, allowing users to receive a verification email with a code after registration, ensuring accurate user identification.

1.5 Database Integration

User authentication and session data are securely stored in the MySQL database. Using Django's ORM system, we efficiently handle user data and enable smooth CRUD operations (Create, Read, Update, Delete).

II.Challenges Faced and Solutions

2.1 Email Delivery Issues

A challenge we encountered was ensuring reliable delivery of verification emails. Initially, some users reported not receiving verification emails, likely due to email client filtering.

Solution: We configured an external SMTP server for handling email delivery and provided clear instructions for users to check their spam or junk mail folders. This significantly improved email delivery rates.

2.2 Session Timeout Management

During testing, we noticed issues with session timeouts, particularly when users were logged in but inactive for extended periods.

Solution: We adjusted the session timeout settings in Django's settings.py, balancing security and user experience. Sessions now expire after 30 minutes of inactivity, but this can be adjusted further based on user feedback.

III.Next Steps

3.1 Blog Post and Comment Features

Next, we will begin developing the core blog features, including post creation, commenting, and search functionality. This will allow users to post blog articles and interact with content in real-time.

3.2 System Testing

We will conduct comprehensive testing of the current user authentication system, including stress tests, to ensure the system can handle concurrent user activity.