# Phase 4 Report: Backend Management and Testing Deployment

Date: November 13, 2024 - December 3, 2024 Group 18

# I.Summary of the Work Done to Date

## 1.1 Backend Management System Development

We utilized Django's built-in admin interface, simplifying backend development while ensuring stability and ease of use for management tasks. Key features include:

**Blog Post Management**: Administrators can view, edit, and delete blog posts through the admin interface, managing the publishing status of posts (draft or published).

**Comment Management**: Comment moderation is handled through the admin interface, where administrators can approve or reject user-submitted comments.

**User Management**: Administrators can manage user accounts through the admin interface, including creating new users, updating information, and disabling accounts.

Additionally, we used Navicat to directly view and manage the database, providing more flexibility for database operations.


## 1.2 Team Member Learning

During this phase, team members learned how to use Django's admin interface, mastering basic operations such as blog post publishing, comment moderation, and user management. This knowledge will be essential for future system maintenance and management.

## 1.3 System Validation and Adjustments

During testing and inspection, we identified some minor issues from earlier stages that prevented certain functionalities from working as expected. To address this, we conducted validation and adjustments:

**Checked and adjusted the view functions** to ensure their logic aligned with expected functionalities.

```python
/blog/pub
@require_http_methods(['GET','POST'])  1 个用法
@login_required(login_url=reverse_lazy('myauth:auth_login'))
def pub_blog(request):
    if request.method =='GET':
        category = BlogCategory.objects.all()
        return render(request, template_name: 'pub_blog.html',context={'category':category})
    else:
        form = PublishBlogForm(request.POST)
        if form.is_valid():
            title = form.cleaned_data.get('title')
            content = form.cleaned_data.get('content')
            category_id = form.cleaned_data.get('category')
            blog = Blog.objects.create(title=title,content=content,category_id=category_id,author=request.user)
            return JsonResponse({'code':200,'message':'发送成功啦！快去首页看看吧','data':{'blog_id':blog.id}})
        else:
            print(form.errors)
            return JsonResponse({"code":200,'message':'发送失败！请检查内容后重新尝试'})
```

**Corrected route mappings** to ensure proper navigation between pages.

**Adjusted the DTL templates** to ensure the layout and functionality of the front-end pages matched the design requirements.

These adjustments ensured that all website functionalities are working as expected.

### 1.4 System Testing

We conducted comprehensive functional and performance tests to ensure the stability of all modules. The tests included:

**Functional Testing**: We thoroughly tested the blog publishing, comment moderation, and user management features to ensure they worked correctly in all expected scenarios.

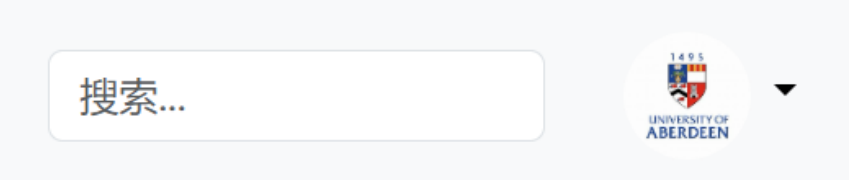
**Performance Testing**: We tested the system's response time and stability under normal usage conditions to ensure it could handle concurrent user access.

### 1.5 System Optimization

Based on the test results, we made several optimizations to improve overall performance and user experience:

**Database Query Optimization**: We added indexes to key queries, improving the performance of the comment and search functions.

**Front-end Optimization**: By compressing and caching static resources (CSS, JS files), we reduced page load times and enhanced the user experience.




## II.Challenges Faced and Solutions

### 2.1 Flexibility of Management Functions

As we relied on Django's admin interface for backend management, the flexibility of some

functions was limited, especially regarding comment moderation and content management.

**Solution**: We supplemented the admin interface with Navicat to directly manage the database, offering more flexibility in data management. This combination addressed the limitations of the admin interface, ensuring efficient data management.

### 2.2 Compatibility Issues During Code Integration

As the project progressed, we encountered compatibility issues between different components, especially when integrating earlier code with new functionalities. Certain features, such as comment moderation or template rendering, did not work as expected due to inconsistencies in the integration process.

**Solution**: We conducted a thorough review of the codebase, identifying and resolving inconsistencies. Adjustments were made to ensure smooth interaction between the components, especially in terms of view functions, routing, and template rendering. This helped ensure that all features operated as intended across the system.

## III.Next Steps

### 3.1 Deployment and Launch

Next, we will complete the project deployment and prepare for the official launch. Before the launch, we will conduct a final round of functional checks and security testing to ensure the system is ready for production.

### 3.2 Continuous Optimization and Improvement

After the system is live, we will continue to collect user feedback and make necessary adjustments to optimize the platform. Our future focus will be on enhancing the user experience and improving the efficiency of backend management.