Package Repository for PoligonVR: A Centralized Asset Management System

Project Documentation

December 20, 2024

Contents

Chapter 1

Introduction and Motivation

1.1 Project Motivation

This Package Repository aims to be an easy-to-use platform for discovering and distributing user-generated content such as unity-packages, 3D models and other assets designed for the social vr platform PoligonVR. By providing a centralized and curated platform, we aim to:

- Reduce time spent searching for assets
- Facilitate community-driven content sharing
- Implement a robust approval mechanism to ensure content quality

Chapter 2

Project Context and Requirements

2.1 Project Purpose

The primary objective of the Package Repository is to create a user-friendly system for:

1. Asset submission
2. Administrative review and approval
3. Comprehensive search and filtering capabilities

2.2 Requirements Analysis: MoSCoW Method

WRITE ABOUT ”PERSOANA” SOWMEWHERE IN REQUIREMENTS

2.2.1 Must Have

- User authentication system
- Package submission functionality
- Admin approval workflow
- Basic search capabilities

2.2.2 Should Have

- Advanced search filtering (by separate title, tags, author)
- Thumbnail preview for packages

2.2.3 Could Have

- Recommendation system
- User ratings for packages
- Price listing for paid assets
- Extra images for submissions

2.2.4 Won't Have

- Fully automated asset validation

Chapter 3

Theoretical Foundations

3.1 Appication Context: Existing Asset Repositories

There are several already existing platforms which fulfill the role of hosting assets. This repository is not meant to be a replacement but to be a unifying layer which sits above these platforms, specifically tailored for use with PoligonVR.

3.1.1 Unity Asset Store

The Unity Asset Store is the main marketplace for unity specific assets:

- Main marketplace for Unity game development assets
- Supports various asset types: 3D models, scripts, textures, animations
- Monetization options for content creators
- Integrated directly with Unity development environment

3.1.2 Sketchfab

A platform focused on 3D content sharing and discovery:

- Supports 3D model uploads and previews
- Supports multiple file format uploads
- Monetization options for premium content

3.1.3 GitHub

GitHub is one of, if not the largest and most diverse code repository:

- Robust version control and tracking
- Vast selection of existing projects
- Integration with continuous integration workflows

3.1.4 Unreal Engine Marketplace

Similar to Unity's approach, focusing on game development assets:

- Marketplace for game development resources
- Support for various asset types
- Direct integration with Unreal Engine
- Monetization for content creators

3.2 Technological Context

This application uses modern web technologies such as:

- Django web framework
- PostgreSQL database
- RESTful architectural principles

Chapter 4

System Implementation

4.1 Architecture Overview

The system adopts the Model-View-Controller (MVC) architecture tailored for scalable web applications. This architectural pattern ensures:

- **Separation of Concerns:** Clear distinction between data management, flow logic, and presentation layers
- **Modularity:** Easy extension and maintenance of individual components
- **Flexibility:** Simplified implementation of new features

4.2 Key Components

4.2.1 User Authentication and Authorization

The authentication system is implemented using Django's built-in authentication framework, providing:

1. **Secure User Registration:**
    - Password hashing using PBKDF2 with a SHA256 hash
    - Enforced password complexity requirements
    - Protection against common vulnerabilities (CSRF, session hijacking)
2. **Role-Based Access Control:**
    - Differentiated access levels (visitor, content creators, administrators)

4.2.2 Application Workflow

The application is based on the K.I.S.S[1] principle and as such it is designed to be as simple to use as possible.

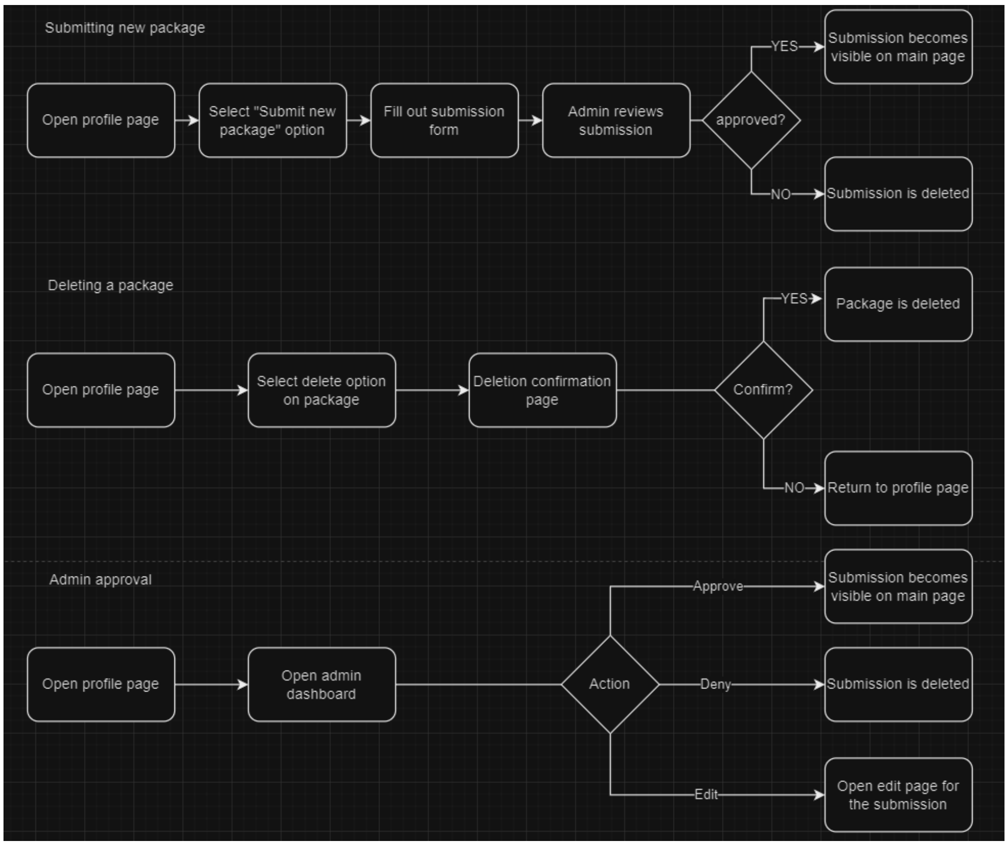

Figure 4.1: Workflow for submitting, deleting and approving

4.2.3 Search and Filtering Mechanism

The search functionality implements advanced querying capabilities:

- **Unified Search:**
  - Simultaneous searching across multiple fields (title, tags, author)
  - Case-insensitive, partial match capabilities
- **Advanced Filtering:**
  - Tag-based filtering

- Author-based search
- Combination of multiple search criteria

4.3 Technical Implementation Details

4.3.1 Database Entity Relationship

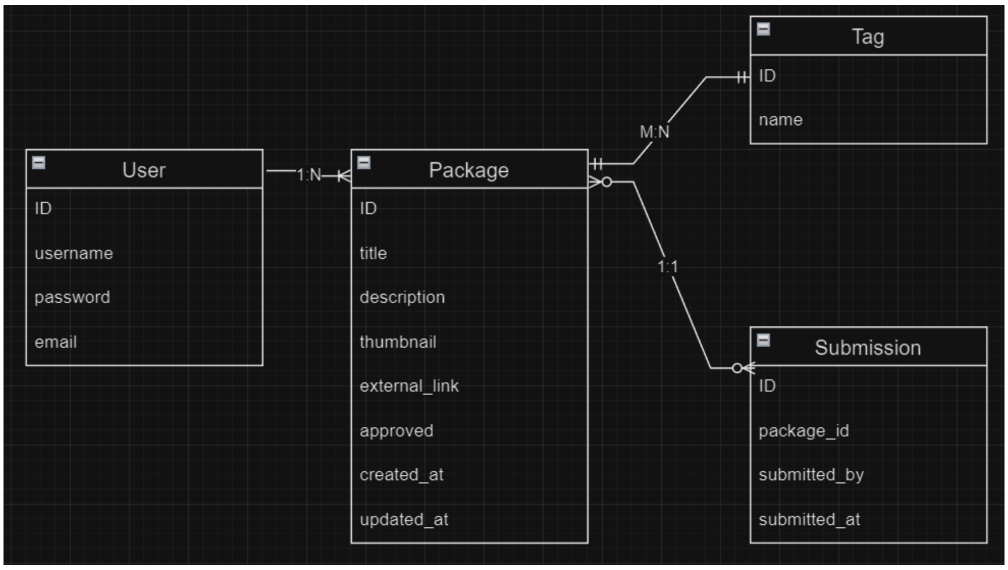

Figure 4.2: Entity-Relationship Diagram for PoligonVR Package Repository

Code for the Package model:

Listing 4.1: Package Definition

```
class Package(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField()
    tags = models.ManyToManyField('Tag')
    thumbnail = models.ImageField(upload_to='thumbnails/')
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    approved = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
```

4.3.2 Image Processing and Validation

Image validation only allows images under 2MB to be uploaded, this ensures low network usage and faster load times:

Listing 4.2: Image Size Validation

```
def validate_image_size(value):
    max_size = 2 * 1048576  # 2MB
    if value.size > max_size:
        raise ValidationError(f'Maximum file size is 2MB')
```

4.3.3 Search Optimization

The search uses Django's built-in ORM[2] for efficient querying:

Listing 4.3: Advanced Search Query

```
def view_submissions(request):
    unified_search = request.GET.get('unified_search', '').strip()
    packages = Package.objects.filter(approved=True)

    if unified_search:
        packages = packages.filter(
            Q(title__icontains=unified_search) |
            Q(author__username__icontains=unified_search) |
            Q(tags__name__icontains=unified_search)
        ).distinct()

    return render(request, 'view_submissions.html', {'packages': package
```

4.3.4 Potential Future Improvements

- **Caching of Search Results:** Implementing server-side caching would reduce database load and improve response times. This can be achieved by temporarily storing frequently accessed search results.
- **Lazy Loading of Thumbnails:** Optimize initial page load times by loading thumbnail images progressively as users scroll. This reduces initial bandwidth consumption and provides a smoother user interface, particularly beneficial for users with slower internet connections.
- **Support for Multiple Images:** Extend the current single-thumbnail model to allow multiple preview images per package. This provides content creators

more comprehensive asset previews and helps users make more informed decisions.

- **Layout Improvements:** Improve the look and feel of the website to make browsing faster and more streamlined. The layout should be adaptable to any screen resolution and mobile devices.
- **Integration support:** Adding support for Sketchfab or Marmoset viewer for displaying 3D assets directly on the page using their embedded viewer. This can be achieved by either automatically detecting links to these platforms or providing a separate link field when submitting an asset.

4.4 Security Implementations

Our security strategy adopts a multi-layered approach to protect both user data and system integrity:

- **CSRF Protection via Django Middleware:** Implements Cross-Site Request Forgery (CSRF) tokens. This prevents malicious websites from performing unauthorized actions on behalf of authenticated users.
- **Role-Based Access Control (RBAC):** Implements a permission systems that:
  - Restrict access to administrative functions
  - Prevent unauthorized package modifications
  - Ensure users can only perform actions appropriate to their role
- **Secure File Upload Handling:** Comprehensive file upload security measures include:
  - File type validation
  - Size restrictions
  - Prevention of potential executable file uploads
- **Additional Security Considerations:**
  - Password complexity enforcement
  - Disallowing commonly used passwords
  - Secure password hashing
  - Protection against common web vulnerabilities

Chapter 5

Testing and Validation

5.1 Test Scenarios

- **Black Box Testing:** All test were primarily conducted using the Black Box method

Scenario	**Tests Conducted**
User Registration	- Validate input constraints - Test password complexity rules - Verify username uniqueness
User Authentication	- Test login with valid/invalid credentials - Verify role-based access control
Package Submission	- Validate form inputs - Test file upload restrictions - Verify admin approval workflow
Search Functionality	- Test unified and advanced search - Validate result relevance
Admin Approval	- Verify package visibility changes - Test approval/rejection workflows

Chapter 6

Conclusion

6.1 Project Achievements

The Package Repository successfully:

- Simplifies asset discovery
- Provided a easy to use platform for content creators

Bibliography

[1] KISS, an acronym for Keep it simple, stupid!, is a design principle first noted by the U.S. Navy in 1960. First seen partly in American English by at least 1938, KISS implies that simplicity should be a design goal. The phrase has been associated with aircraft engineer Kelly Johnson.

[2] The Django ORM is an implementation of the object-relational mapping (ORM) concept.