Django Backend Architecture Design

Current Analysis

Existing Django Structure

The current backend is a Django application with:
- **Core App**: Contains user management, task processing, and LLM integration
- **Models**: User (extended AbstractUser), Task, Package, CreditTransaction, PromptTemplate
- **Views**: Authentication, user management, task processing with streaming responses
- **LLM Integration**: OpenAI integration for paper generation
- **Static Prompts**: Hardcoded prompts in files for different paper types (paper, proposal, report)

Frontend Static Content to Make Dynamic

From the frontend analysis, the following static content should be made admin-manageable:
- **Landing Page Content**: Hero section, features, testimonials, CTA sections
- **Pricing Plans**: Package details, features, pricing tiers
- **Templates**: Paper templates and formats
- **Localization**: Multi-language content (en, zh-CN, zh-TW, ja, ko, ha)

New Django Architecture Design

1. App Structure Reorganization

```
academic_paper_generator/
├── config/                 # Django settings and configuration
│   ├── __init__.py
│   ├── settings/
│   │   ├── __init__.py
│   │   ├── base.py         # Base settings
│   │   ├── development.py  # Development settings
│   │   ├── production.py   # Production settings
│   │   └── testing.py      # Testing settings
│   ├── urls.py
│   ├── wsgi.py
```

```
│   └── asgi.py
├── apps/
│   ├── __init__.py
│   ├── users/              # User management
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── serializers.py
│   │   ├── admin.py
│   │   └── urls.py
│   ├── content/            # CMS-like content management
│   │   ├── models.py       # Landing page, testimonials, etc.
│   │   ├── views.py
│   │   ├── serializers.py
│   │   ├── admin.py
│   │   └── urls.py
│   ├── papers/             # Paper generation and templates
│   │   ├── models.py       # Paper formats, templates
│   │   ├── views.py
│   │   ├── serializers.py
│   │   ├── admin.py
│   │   ├── generators/     # Paper generation logic
│   │   └── urls.py
│   ├── billing/            # Credits and packages
│   │   ├── models.py
│   │   ├── views.py
│   │   ├── serializers.py
│   │   ├── admin.py
│   │   └── urls.py
│   └── core/               # Shared utilities
│       ├── models.py       # Abstract base models
│       ├── permissions.py
│       ├── utils.py
│       └── mixins.py
├── static/
├── media/
├── templates/
├── requirements/
│   ├── base.txt
│   ├── development.txt
│   └── production.txt
├── manage.py
└── README.md
```

2. New Model Design

Content Management Models

```python
# apps/content/models.py
class LandingPageSection(models.Model):
    """Admin-manageable landing page sections"""
    SECTION_TYPES = [
        ('hero', 'Hero Section'),
        ('features', 'Features Section'),
        ('testimonials', 'Testimonials Section'),
        ('cta', 'Call to Action Section'),
    ]

    section_type = models.CharField(max_length=20, choices=SECTION_TYPES)
    language = models.CharField(max_length=10, default='en')
    title = models.CharField(max_length=200)
    subtitle = models.TextField(blank=True)
    content = models.JSONField()  # Flexible content storage
    is_active = models.BooleanField(default=True)
    order = models.PositiveIntegerField(default=0)

class Testimonial(models.Model):
    """User testimonials"""
    name = models.CharField(max_length=100)
    role = models.CharField(max_length=100)
    company = models.CharField(max_length=100, blank=True)
    content = models.TextField()
    avatar = models.ImageField(upload_to='testimonials/', blank=True)
    rating = models.PositiveIntegerField(default=5)
    language = models.CharField(max_length=10, default='en')
    is_featured = models.BooleanField(default=False)
```

Paper Management Models

```python
# apps/papers/models.py
class PaperFormat(models.Model):
    """Different academic paper formats"""
    name = models.CharField(max_length=100)
    description = models.TextField()
    template_structure = models.JSONField()  # JSON structure for paper sections
    style_guidelines = models.TextField()
    is_active = models.BooleanField(default=True)

class PaperTemplate(models.Model):
    """Specific paper templates"""
    name = models.CharField(max_length=100)
    format = models.ForeignKey(PaperFormat, on_delete=models.CASCADE)
    language = models.CharField(max_length=10, default='en')
```

```
    system_prompt = models.TextField()
    user_prompt_template = models.TextField()
    example_output = models.TextField(blank=True)

class GeneratedPaper(models.Model):
    """Generated papers history"""
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    template = models.ForeignKey(PaperTemplate, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    content = models.TextField()
    parameters = models.JSONField()  # User inputs
    status = models.CharField(max_length=20, default='completed')
    created_at = models.DateTimeField(auto_now_add=True)
```

Enhanced User and Billing Models

```
# apps/users/models.py - Enhanced User model
class UserProfile(models.Model):
    """Extended user profile"""
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    avatar = models.ImageField(upload_to='avatars/', blank=True)
    bio = models.TextField(blank=True)
    institution = models.CharField(max_length=200, blank=True)
    field_of_study = models.CharField(max_length=100, blank=True)

# apps/billing/models.py
class Package(models.Model):
    """Restructured package model"""
    name = models.CharField(max_length=100)
    description = models.TextField()
    credits = models.PositiveIntegerField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    currency = models.CharField(max_length=3, default='USD')
    features = models.JSONField()  # List of features
    is_popular = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    order = models.PositiveIntegerField(default=0)
```

3. API Design

RESTful API Structure

```
/api/v1/
├── auth/                    # Authentication endpoints
│   ├── login/
│   ├── register/
│   ├── refresh/
│   └── logout/
```

```
├── users/                  # User management
│   ├── profile/
│   ├── stats/
│   └── preferences/
├── content/                # CMS content
│   ├── landing-sections/
│   ├── testimonials/
│   └── localization/
├── papers/                 # Paper generation
│   ├── formats/
│   ├── templates/
│   ├── generate/
│   └── history/
├── billing/                # Billing and packages
│   ├── packages/
│   ├── transactions/
│   └── credits/
└── admin/                  # Admin-specific endpoints
    ├── analytics/
    └── system-status/
```

4. Admin Interface Design

Custom Admin Configuration

- **Dashboard**: Overview with key metrics, recent activities
- **Content Management**: Landing page sections, testimonials, localization
- **Paper Management**: Formats, templates, generation history
- **User Management**: Enhanced user profiles, activity tracking
- **Billing Management**: Packages, transactions, credit management
- **System Management**: API usage, performance metrics

5. Paper Generation Module

Template-Based Generation System

```python
class PaperGenerator:
    """Main paper generation class"""

    def __init__(self, template: PaperTemplate):
        self.template = template

    def generate(self, user_inputs: dict) -> str:
        """Generate paper based on template and user inputs"""
        # 1. Validate inputs against template requirements
        # 2. Build prompts from template
```

```
        # 3. Call LLM service
        # 4. Format output according to paper format
        # 5. Save to database

    def stream_generate(self, user_inputs: dict):
        """Stream generation for real-time updates"""
        # Similar to generate but with streaming response
```

6. Enhanced Security and Performance

Security Improvements

- Rate limiting for API endpoints
- Enhanced authentication with JWT refresh tokens
- Input validation and sanitization
- CORS configuration for frontend integration

Performance Optimizations

- Database query optimization with select_related/prefetch_related
- Caching for frequently accessed content
- Asynchronous task processing for paper generation
- API response compression

7. Documentation and Testing

API Documentation

- Complete Swagger/OpenAPI documentation
- Interactive API explorer
- Code examples for all endpoints
- Authentication flow documentation

Testing Strategy

- Unit tests for all models and business logic
- Integration tests for API endpoints
- Performance tests for paper generation
- Security tests for authentication and authorization

This architecture provides:
1. **Scalability**: Modular app structure allows for easy expansion
2. **Maintainability**: Clear separation of concerns and Django best practices
3. **Flexibility**: JSON fields for dynamic content and template structures

4. **Admin-Friendly**: Comprehensive admin interface for content management
5. **API-First**: Well-designed RESTful API with complete documentation
6. **Performance**: Optimized for both development and production environments