🛡️ Complete Admin Site Implementation Guide

Phase 1: Database & Backend Foundation (Week 1)

Step 1: Update User Model for Admin Functionality

```
python
```

```python
# backend/models/user.py - Add these fields to your existing User model

class User(db.Model):
    # ... existing fields ...

    # Admin fields
    is_admin = db.Column(db.Boolean, default=False)
    admin_level = db.Column(db.String(50), nullable=True)  # 'super_admin', 'moderator', 'manag
    admin_permissions = db.Column(db.JSON, nullable=True)
    admin_notes = db.Column(db.Text, nullable=True)
    admin_created_by = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    admin_created_at = db.Column(db.DateTime, nullable=True)

    # User management fields
    banned = db.Column(db.Boolean, default=False)
    ban_reason = db.Column(db.String(500), nullable=True)
    banned_by = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    banned_at = db.Column(db.DateTime, nullable=True)
    last_login = db.Column(db.DateTime, nullable=True)

    # Relationships for admin tracking
    created_admins = db.relationship('User',
                                     foreign_keys=[admin_created_by],
                                     remote_side=[id])
    banned_users = db.relationship('User',
                                   foreign_keys=[banned_by],
                                   remote_side=[id])

    def has_admin_permission(self, permission):
        """Check if admin has specific permission"""
        if not self.is_admin or not self.admin_permissions:
            return False
        return permission in self.admin_permissions

    def to_admin_dict(self):
        """Return user data for admin views (includes more fields)"""
        return {
            'id': self.id,
            'name': self.name,
            'email': self.email,
            'university': self.university,
            'year_of_study': self.year_of_study,
            'phone': self.phone,
```

```
            'is_admin': self.is_admin,
            'admin_level': self.admin_level,
            'banned': self.banned,
            'ban_reason': self.ban_reason,
            'created_at': self.created_at.isoformat() if self.created_at else None,
            'last_login': self.last_login.isoformat() if self.last_login else None,
            'total_reviews': len(self.reviews),
            'total_applications': len([app for app in self.applications])
        }
```

Step 2: Create Admin-Specific Models

```
python
```

```python
# backend/models/admin.py - Create new file

from datetime import datetime
from app import db

class AdminLog(db.Model):
    """Track all admin actions for audit trail"""
    __tablename__ = 'admin_logs'

    id = db.Column(db.Integer, primary_key=True)
    admin_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    action = db.Column(db.String(100), nullable=False)  # 'ban_user', 'approve_property', etc.
    target_type = db.Column(db.String(50), nullable=False)  # 'user', 'property', 'review'
    target_id = db.Column(db.Integer, nullable=False)
    details = db.Column(db.JSON, nullable=True)  # Store additional context
    ip_address = db.Column(db.String(45), nullable=True)
    user_agent = db.Column(db.String(500), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    # Relationships
    admin = db.relationship('User', backref='admin_actions')

    def __repr__(self):
        return f'<AdminLog {self.admin_id}: {self.action} on {self.target_type}#{self.target_id}>'

class PropertyApproval(db.Model):
    """Track property approval status and workflow"""
    __tablename__ = 'property_approvals'

    id = db.Column(db.Integer, primary_key=True)
    property_id = db.Column(db.Integer, db.ForeignKey('property.id'), nullable=False)
    status = db.Column(db.String(20), default='pending')  # 'pending', 'approved', 'rejected',
    submitted_at = db.Column(db.DateTime, default=datetime.utcnow)
    reviewed_by = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    reviewed_at = db.Column(db.DateTime, nullable=True)
    admin_notes = db.Column(db.Text, nullable=True)
    rejection_reason = db.Column(db.Text, nullable=True)

    # Relationships
    property = db.relationship('Property', backref='approval_status')
    reviewer = db.relationship('User')

    def __repr__(self):
```

```
        return f'<PropertyApproval {self.property_id}: {self.status}>'

class SystemSettings(db.Model):
    """Store admin-configurable system settings"""
    __tablename__ = 'system_settings'

    id = db.Column(db.Integer, primary_key=True)
    key = db.Column(db.String(100), unique=True, nullable=False)
    value = db.Column(db.Text, nullable=True)
    description = db.Column(db.Text, nullable=True)
    updated_by = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow)

    # Relationships
    updater = db.relationship('User')
```

Step 3: Create Database Migration

bash

```
# Run these commands in your backend directory
flask db migrate -m "Add admin functionality and approval system"
flask db upgrade
```

Step 4: Create First Admin User

python

```python
# backend/create_admin.py - Script to create first admin

from app import app, db
from models.user import User
from datetime import datetime
import bcrypt

def create_admin():
    with app.app_context():
        # Check if admin exists
        admin = User.query.filter_by(email='admin@yourdomain.com').first()

        if not admin:
            # Create admin user
            admin = User(
                name='Admin User',
                email='admin@yourdomain.com',
                password=bcrypt.hashpw('admin123'.encode('utf-8'), bcrypt.gensalt()).decode('ut
                university='admin',
                year_of_study='N/A',
                is_admin=True,
                admin_level='super_admin',
                admin_permissions=['manage_users', 'approve_properties', 'moderate_reviews', 'v
                admin_created_at=datetime.utcnow(),
                verified=True
            )

            db.session.add(admin)
            db.session.commit()
            print("Admin user created successfully!")
            print("Email: admin@yourdomain.com")
            print("Password: admin123")
            print("Please change the password after first login!")
        else:
            print("Admin user already exists")

if __name__ == '__main__':
    create_admin()
```

Phase 2: Admin Authentication & Middleware (Week 1-2)

Step 5: Create Admin Authentication Middleware

```
python
```

```python
# backend/utils/admin_decorators.py - Create new file

from functools import wraps
from flask import jsonify, request
from flask_jwt_extended import verify_jwt_in_request, get_jwt_identity
from models.user import User
from models.admin import AdminLog
import requests

def admin_required(permission=None):
    """Decorator to require admin access with optional specific permission"""
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            # Verify JWT token
            verify_jwt_in_request()
            current_user_id = get_jwt_identity()

            # Get user
            user = User.query.get(current_user_id)
            if not user:
                return jsonify({'error': 'User not found'}), 404

            # Check if user is admin
            if not user.is_admin:
                return jsonify({'error': 'Admin access required'}), 403

            # Check specific permission if required
            if permission and not user.has_admin_permission(permission):
                return jsonify({'error': f'Permission {permission} required'}), 403

            # Set current admin in request context
            request.current_admin = user

            return f(*args, **kwargs)
        return decorated_function
    return decorator

def log_admin_action(action, target_type, target_id, details=None):
    """Log admin action for audit trail"""
    try:
        admin_log = AdminLog(
            admin_id=request.current_admin.id,
```

```
            action=action,
            target_type=target_type,
            target_id=target_id,
            details=details,
            ip_address=request.remote_addr,
            user_agent=request.headers.get('User-Agent', '')
        )

        db.session.add(admin_log)
        db.session.commit()
    except Exception as e:
        print(f"Failed to log admin action: {e}")
```

Step 6: Admin Authentication Routes

```
python
```

```python
# backend/routes/admin_auth.py - Create new file

from flask import Blueprint, request, jsonify
from flask_jwt_extended import jwt_required, get_jwt_identity, create_access_token
from models.user import User
from models.admin import AdminLog
from utils.admin_decorators import admin_required, log_admin_action
from datetime import datetime

admin_auth_bp = Blueprint('admin_auth', __name__)

@admin_auth_bp.route('/admin/login', methods=['POST'])
def admin_login():
    """Admin login endpoint with enhanced security"""
    try:
        data = request.get_json()
        email = data.get('email')
        password = data.get('password')

        if not email or not password:
            return jsonify({'error': 'Email and password required'}), 400

        # Find user
        user = User.query.filter_by(email=email).first()

        if not user or not user.check_password(password):
            # Log failed login attempt
            log_failed_login(email, request.remote_addr)
            return jsonify({'error': 'Invalid credentials'}), 401

        # Check if user is admin
        if not user.is_admin:
            return jsonify({'error': 'Admin access required'}), 403

        # Check if admin is banned
        if user.banned:
            return jsonify({'error': 'Account suspended'}), 403

        # Update last login
        user.last_login = datetime.utcnow()
        db.session.commit()

        # Create JWT token with admin claims
```

```
        additional_claims = {
            'is_admin': True,
            'admin_level': user.admin_level,
            'permissions': user.admin_permissions or []
        }

        access_token = create_access_token(
            identity=user.id,
            additional_claims=additional_claims
        )

        # Log successful admin login
        log_admin_action('admin_login', 'user', user.id, {
            'login_time': datetime.utcnow().isoformat(),
            'ip_address': request.remote_addr
        })

        return jsonify({
            'access_token': access_token,
            'admin': {
                'id': user.id,
                'name': user.name,
                'email': user.email,
                'admin_level': user.admin_level,
                'permissions': user.admin_permissions
            }
        }), 200

    except Exception as e:
        return jsonify({'error': 'Login failed'}), 500

@admin_auth_bp.route('/admin/profile', methods=['GET'])
@admin_required()
def get_admin_profile():
    """Get current admin profile"""
    admin = request.current_admin

    return jsonify({
        'admin': {
            'id': admin.id,
            'name': admin.name,
            'email': admin.email,
            'admin_level': admin.admin_level,
            'permissions': admin.admin_permissions,
```

```
            'created_at': admin.admin_created_at.isoformat() if admin.admin_created_at else Nor
            'last_login': admin.last_login.isoformat() if admin.last_login else None
        }
    }), 200

def log_failed_login(email, ip_address):
    """Log failed admin login attempts"""
    try:
        # You could store this in a separate table for security monitoring
        print(f"Failed admin login attempt: {email} from {ip_address}")
    except Exception as e:
        print(f"Failed to log failed login: {e}")
```

Phase 3: Core Admin APIs (Week 2)

Step 7: User Management APIs

```
python
```

```python
# backend/routes/admin_users.py - Create new file

from flask import Blueprint, request, jsonify
from flask_jwt_extended import jwt_required
from models.user import User
from models.admin import AdminLog
from utils.admin_decorators import admin_required, log_admin_action
from datetime import datetime

admin_users_bp = Blueprint('admin_users', __name__)

@admin_users_bp.route('/admin/users', methods=['GET'])
@admin_required('manage_users')
def get_all_users():
    """Get all users with filtering and pagination"""
    try:
        # Parse query parameters
        page = request.args.get('page', 1, type=int)
        per_page = request.args.get('per_page', 50, type=int)
        university = request.args.get('university')
        banned = request.args.get('banned', type=bool)
        search = request.args.get('search')
        admin_only = request.args.get('admin_only', type=bool)

        # Build query
        query = User.query

        # Apply filters
        if university:
            query = query.filter(User.university == university)
        if banned is not None:
            query = query.filter(User.banned == banned)
        if admin_only:
            query = query.filter(User.is_admin == True)
        if search:
            query = query.filter(
                User.name.contains(search) |
                User.email.contains(search)
            )

        # Order by newest first
        query = query.order_by(User.created_at.desc())
```

```
        # Paginate
        users_pagination = query.paginate(
            page=page,
            per_page=per_page,
            error_out=False
        )

        # Log admin action
        log_admin_action('view_users', 'user', 0, {
            'filters': {
                'university': university,
                'banned': banned,
                'search': search,
                'admin_only': admin_only
            },
            'page': page
        })

        return jsonify({
            'users': [user.to_admin_dict() for user in users_pagination.items],
            'total': users_pagination.total,
            'pages': users_pagination.pages,
            'current_page': page,
            'per_page': per_page
        }), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch users'}), 500

@admin_users_bp.route('/admin/users/<int:user_id>', methods=['GET'])
@admin_required('manage_users')
def get_user_details(user_id):
    """Get detailed information about a specific user"""
    try:
        user = User.query.get_or_404(user_id)

        # Get additional user statistics
        user_data = user.to_admin_dict()
        user_data.update({
            'reviews': [review.to_dict() for review in user.reviews[-5:]],  # Last 5 reviews
            'applications': len(user.applications),
            'admin_notes': user.admin_notes,
            'ban_history': get_user_ban_history(user_id)
        })
```

```
        # Log admin action
        log_admin_action('view_user_details', 'user', user_id)

        return jsonify({'user': user_data}), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch user details'}), 500

@admin_users_bp.route('/admin/users/<int:user_id>/ban', methods=['POST'])
@admin_required('manage_users')
def ban_user(user_id):
    """Ban or unban a user"""
    try:
        user = User.query.get_or_404(user_id)
        data = request.get_json()

        action = data.get('action')  # 'ban' or 'unban'
        reason = data.get('reason', '')

        if action == 'ban':
            user.banned = True
            user.ban_reason = reason
            user.banned_by = request.current_admin.id
            user.banned_at = datetime.utcnow()

            log_admin_action('ban_user', 'user', user_id, {
                'reason': reason,
                'banned_by': request.current_admin.name
            })

            message = f'User {user.name} has been banned'

        elif action == 'unban':
            user.banned = False
            user.ban_reason = None
            user.banned_by = None
            user.banned_at = None

            log_admin_action('unban_user', 'user', user_id, {
                'unbanned_by': request.current_admin.name
            })

            message = f'User {user.name} has been unbanned'
```

```
        else:
            return jsonify({'error': 'Action must be ban or unban'}), 400

        db.session.commit()

        return jsonify({
            'message': message,
            'user': user.to_admin_dict()
        }), 200

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': 'Failed to update user ban status'}), 500

@admin_users_bp.route('/admin/users/<int:user_id>/make-admin', methods=['POST'])
@admin_required('manage_admins')
def make_user_admin(user_id):
    """Promote user to admin or modify admin permissions"""
    try:
        user = User.query.get_or_404(user_id)
        data = request.get_json()

        admin_level = data.get('admin_level', 'moderator')
        permissions = data.get('permissions', ['manage_users', 'approve_properties'])

        # Validate admin level
        valid_levels = ['moderator', 'manager', 'super_admin']
        if admin_level not in valid_levels:
            return jsonify({'error': 'Invalid admin level'}), 400

        # Update user to admin
        user.is_admin = True
        user.admin_level = admin_level
        user.admin_permissions = permissions
        user.admin_created_by = request.current_admin.id
        user.admin_created_at = datetime.utcnow()

        db.session.commit()

        # Log admin action
        log_admin_action('promote_to_admin', 'user', user_id, {
            'admin_level': admin_level,
            'permissions': permissions,
```

```
            'promoted_by': request.current_admin.name
        })

        return jsonify({
            'message': f'{user.name} has been promoted to admin',
            'user': user.to_admin_dict()
        }), 200

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': 'Failed to promote user to admin'}), 500

def get_user_ban_history(user_id):
    """Get ban history for a user"""
    ban_logs = AdminLog.query.filter(
        AdminLog.target_type == 'user',
        AdminLog.target_id == user_id,
        AdminLog.action.in_(['ban_user', 'unban_user'])
    ).order_by(AdminLog.created_at.desc()).limit(10).all()

    return [{
        'action': log.action,
        'admin_name': log.admin.name,
        'date': log.created_at.isoformat(),
        'details': log.details
    } for log in ban_logs]
```

Step 8: Property Management APIs

```
python
```

```python
# backend/routes/admin_properties.py - Create new file

from flask import Blueprint, request, jsonify
from models.property import Property
from models.admin import PropertyApproval, AdminLog
from utils.admin_decorators import admin_required, log_admin_action
from datetime import datetime

admin_properties_bp = Blueprint('admin_properties', __name__)

@admin_properties_bp.route('/admin/properties/pending', methods=['GET'])
@admin_required('approve_properties')
def get_pending_properties():
    """Get properties awaiting approval"""
    try:
        page = request.args.get('page', 1, type=int)
        per_page = request.args.get('per_page', 20, type=int)

        # Get properties with pending approval status
        pending_query = db.session.query(Property, PropertyApproval).join(
            PropertyApproval, Property.id == PropertyApproval.property_id
        ).filter(PropertyApproval.status == 'pending').order_by(
            PropertyApproval.submitted_at.asc()
        )

        pending_pagination = pending_query.paginate(
            page=page,
            per_page=per_page,
            error_out=False
        )

        properties_data = []
        for property_obj, approval in pending_pagination.items:
            property_dict = property_obj.to_dict()
            property_dict.update({
                'approval_id': approval.id,
                'submitted_at': approval.submitted_at.isoformat(),
                'days_pending': (datetime.utcnow() - approval.submitted_at).days
            })
            properties_data.append(property_dict)

        # Log admin action
        log_admin_action('view_pending_properties', 'property', 0, {
```

```
            'page': page,
            'count': len(properties_data)
        })

        return jsonify({
            'properties': properties_data,
            'total': pending_pagination.total,
            'pages': pending_pagination.pages,
            'current_page': page
        }), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch pending properties'}), 500

@admin_properties_bp.route('/admin/properties/<int:property_id>/approve', methods=['POST'])
@admin_required('approve_properties')
def approve_property(property_id):
    """Approve or reject a property"""
    try:
        data = request.get_json()
        action = data.get('action')  # 'approve' or 'reject'
        notes = data.get('notes', '')

        property_obj = Property.query.get_or_404(property_id)
        approval = PropertyApproval.query.filter_by(property_id=property_id).first()

        if not approval:
            return jsonify({'error': 'Property approval record not found'}), 404

        if action == 'approve':
            approval.status = 'approved'
            property_obj.approved = True
            property_obj.approved_at = datetime.utcnow()
            message = f'Property "{property_obj.name}" has been approved'

        elif action == 'reject':
            approval.status = 'rejected'
            approval.rejection_reason = notes
            property_obj.approved = False
            message = f'Property "{property_obj.name}" has been rejected'

        else:
            return jsonify({'error': 'Action must be approve or reject'}), 400
```

```
        approval.reviewed_by = request.current_admin.id
        approval.reviewed_at = datetime.utcnow()
        approval.admin_notes = notes

        db.session.commit()

        # Log admin action
        log_admin_action(f'{action}_property', 'property', property_id, {
            'property_name': property_obj.name,
            'notes': notes,
            'reviewed_by': request.current_admin.name
        })

        return jsonify({
            'message': message,
            'property': property_obj.to_dict()
        }), 200

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': f'Failed to {action} property'}), 500

@admin_properties_bp.route('/admin/properties', methods=['GET'])
@admin_required('approve_properties')
def get_all_properties():
    """Get all properties with admin information"""
    try:
        page = request.args.get('page', 1, type=int)
        per_page = request.args.get('per_page', 50, type=int)
        status = request.args.get('status')  # 'approved', 'pending', 'rejected'
        university = request.args.get('university')
        search = request.args.get('search')

        # Build query
        query = Property.query

        # Apply filters
        if status:
            if status == 'pending':
                query = query.join(PropertyApproval).filter(PropertyApproval.status == 'pending
            elif status == 'approved':
                query = query.filter(Property.approved == True)
            elif status == 'rejected':
                query = query.join(PropertyApproval).filter(PropertyApproval.status == 'rejecte
```

```
        if university:
            query = query.filter(Property.university == university)

        if search:
            query = query.filter(
                Property.name.contains(search) |
                Property.address.contains(search)
            )

        # Order by newest first
        query = query.order_by(Property.created_at.desc())

        # Paginate
        properties_pagination = query.paginate(
            page=page,
            per_page=per_page,
            error_out=False
        )

        # Get properties with approval status
        properties_data = []
        for prop in properties_pagination.items:
            property_dict = prop.to_dict()

            # Get approval status
            approval = PropertyApproval.query.filter_by(property_id=prop.id).first()
            if approval:
                property_dict.update({
                    'approval_status': approval.status,
                    'submitted_at': approval.submitted_at.isoformat(),
                    'reviewed_at': approval.reviewed_at.isoformat() if approval.reviewed_at els
                    'admin_notes': approval.admin_notes
                })

            properties_data.append(property_dict)

        # Log admin action
        log_admin_action('view_properties', 'property', 0, {
            'filters': {
                'status': status,
                'university': university,
                'search': search
            },
```

```
            'page': page
        })

        return jsonify({
            'properties': properties_data,
            'total': properties_pagination.total,
            'pages': properties_pagination.pages,
            'current_page': page
        }), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch properties'}), 500
```

Phase 4: Frontend Admin Interface (Week 3)

Step 9: Admin Route Protection & Layout

jsx

```jsx
// frontend/src/components/AdminRoute.js
import React from 'react';
import { Navigate } from 'react-router-dom';
import { useAuth } from '../contexts/AuthContext';

const AdminRoute = ({ children, requiredPermission = null }) => {
  const { user, isAuthenticated, loading } = useAuth();

  if (loading) {
    return <div className="min-h-screen flex items-center justify-center">
      <div className="animate-spin rounded-full h-32 w-32 border-b-2 border-blue-600"></div>
    </div>;
  }

  if (!isAuthenticated) {
    return <Navigate to="/admin/login" replace />;
  }

  if (!user?.is_admin) {
    return <Navigate to="/dashboard" replace />;
  }

  if (requiredPermission && !user?.permissions?.includes(requiredPermission)) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <h1 className="text-2xl font-bold text-gray-900 mb-4">Access Denied</h1>
          <p className="text-gray-600">You don't have permission to access this page.</p>
        </div>
      </div>
    );
  }

  return children;
};

export default AdminRoute;
```

jsx

```
// frontend/src/components/Admin/AdminLayout.js
import React, { useState } from 'react';
import { Link, useLocation, useNavigate } from 'react-router-dom';
import {
  Users,
  Home,
  Settings,
  LogOut,
  Shield,
  BarChart3,
  MessageSquare,
  Menu,
  X
} from 'lucide-react';
import { useAuth } from '../../contexts/AuthContext';

const AdminLayout = ({ children }) => {
  const [sidebarOpen, setSidebarOpen] = useState(false);
  const location = useLocation();
  const navigate = useNavigate();
  const { user, logout } = useAuth();

  const navigation = [
    { name: 'Dashboard', href: '/admin', icon: Home, permission: null },
    { name: 'Users', href: '/admin/users', icon: Users, permission: 'manage_users' },
    { name: 'Properties', href: '/admin/properties', icon: Shield, permission: 'approve_propert
    { name: 'Reviews', href: '/admin/reviews', icon: MessageSquare, permission: 'moderate_revie
    { name: 'Analytics', href: '/admin/analytics', icon: BarChart3, permission: 'view_analytics
    { name: 'Settings', href: '/admin/settings', icon: Settings, permission: 'manage_settings'
  ];

  const handleLogout = () => {
    logout();
    navigate('/admin/login');
  };

  const hasPermission = (permission) => {
    if (!permission) return true;
    return user?.permissions?.includes(permission);
  };

  return (
    <div className="min-h-screen bg-gray-50 flex">
```

```
      {/* Mobile sidebar overlay */}
      {sidebarOpen && (
        <div className="fixed inset-0 z-40 lg:hidden">
          <div className="fixed inset-0 bg-gray-600 bg-opacity-75" onClick={() => setSidebarOpe
        </div>
      )}

      {/* Sidebar */}
      <div className={`fixed inset-y-0 left-0 z-50 w-64 bg-white shadow-lg transform transitior
        sidebarOpen ? 'translate-x-0' : '-translate-x-full'
      }`}>
        <div className="flex items-center justify-between h-16 px-6 border-b border-gray-200">
          <h1 className="text-xl font-bold text-gray-900">Admin Panel</h1>
          <button
            onClick={() => setSidebarOpen(false)}
            className="lg:hidden text-gray-400 hover:text-gray-600"
          >
            <X className="w-6 h-6" />
          </button>
        </div>

        <nav className="mt-6 px-3">
          <div className="space-y-1">
            {navigation.map((item) => {
              if (!hasPermission(item.permission)) return null;

              const isActive = location.pathname === item.href;
              return (
                <Link
                  key={item.name}
                  to={item.href}
                  className={`group flex items-center px-3 py-2 text-sm font-medium rounded-md
                    isActive
                      ? 'bg-blue-100 text-blue-700'
                      : 'text-gray-600 hover:bg-gray-50 hover:text-gray-900'
                  }`}
                >
                  <item.icon
                    className={`mr-3 h-5 w-5 ${
                      isActive ? 'text-blue-500' : 'text-gray-400 group-hover:text-gray-500'
                    }`}
                  />
                  {item.name}
                </Link>
```

```
            );
          })}
        </div>
      </nav>

      {/* Admin info */}
      <div className="absolute bottom-0 left-0 right-0 p-4 border-t border-gray-200">
        <div className="flex items-center">
          <div className="flex-1">
            <p className="text-sm font-medium text-gray-900">{user?.name}</p>
            <p className="text-xs text-gray-500 capitalize">{user?.admin_level}</p>
          </div>
          <button
            onClick={handleLogout}
            className="text-gray-400 hover:text-gray-600"
            title="Logout"
          >
            <LogOut className="w-5 h-5" />
          </button>
        </div>
      </div>
    </div>

    {/* Main content */}
    <div className="flex-1 flex flex-col overflow-hidden">
      {/* Top bar */}
      <header className="bg-white shadow-sm border-b border-gray-200">
        <div className="flex items-center justify-between h-16 px-6">
          <button
            onClick={() => setSidebarOpen(true)}
            className="lg:hidden text-gray-400 hover:text-gray-600"
          >
            <Menu className="w-6 h-6" />
          </button>

          <div className="flex items-center space-x-4">
            <span className="text-sm text-gray-500">
              Last login: {user?.last_login ? new Date(user.last_login).toLocaleString() : 'N
            </span>
          </div>
        </div>
      </header>

      {/* Page content */}
```

```
        <main className="flex-1 overflow-auto">
          <div className="max-w-7xl mx-auto px-6 py-8">
            {children}
          </div>
        </main>
      </div>
    </div>
  );
};

export default AdminLayout;
```

Step 10: Admin Dashboard

jsx

```
// frontend/src/pages/Admin/AdminDashboard.js
import React, { useState, useEffect } from 'react';
import {
  Users,
  Home,
  MessageSquare,
  TrendingUp,
  UserPlus,
  CheckCircle,
  AlertCircle,
  Clock
} from 'lucide-react';

const AdminDashboard = () => {
  const [stats, setStats] = useState({
    totalUsers: 0,
    totalProperties: 0,
    totalReviews: 0,
    pendingApprovals: 0,
    newUsersThisWeek: 0,
    approvedPropertiesThisWeek: 0,
    flaggedReviews: 0,
    activeUsers: 0
  });
  const [recentActivity, setRecentActivity] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchDashboardData();
  }, []);

  const fetchDashboardData = async () => {
    try {
      setLoading(true);

      // Fetch dashboard statistics
      const statsResponse = await fetch('/api/admin/dashboard/stats', {
        headers: {
          'Authorization': `Bearer ${localStorage.getItem('token')}`
        }
      });

      if (statsResponse.ok) {
```

```
        const statsData = await statsResponse.json();
        setStats(statsData);
      }

      // Fetch recent activity
      const activityResponse = await fetch('/api/admin/dashboard/recent-activity', {
        headers: {
          'Authorization': `Bearer ${localStorage.getItem('token')}`
        }
      });

      if (activityResponse.ok) {
        const activityData = await activityResponse.json();
        setRecentActivity(activityData.activities);
      }

    } catch (error) {
      console.error('Error fetching dashboard data:', error);
    } finally {
      setLoading(false);
    }
  };

  const statCards = [
    {
      title: 'Total Users',
      value: stats.totalUsers,
      icon: Users,
      color: 'bg-blue-500',
      change: `+${stats.newUsersThisWeek} this week`
    },
    {
      title: 'Total Properties',
      value: stats.totalProperties,
      icon: Home,
      color: 'bg-green-500',
      change: `+${stats.approvedPropertiesThisWeek} approved this week`
    },
    {
      title: 'Total Reviews',
      value: stats.totalReviews,
      icon: MessageSquare,
      color: 'bg-purple-500',
      change: `${stats.flaggedReviews} flagged`
```

```
    },
    {
      title: 'Pending Approvals',
      value: stats.pendingApprovals,
      icon: Clock,
      color: 'bg-orange-500',
      change: 'Require attention'
    }
  ];

  if (loading) {
    return (
      <div className="animate-pulse space-y-6">
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
          {[...Array(4)].map((_, i) => (
            <div key={i} className="bg-gray-200 rounded-lg h-32"></div>
          ))}
        </div>
        <div className="bg-gray-200 rounded-lg h-96"></div>
      </div>
    );
  }

  return (
    <div className="space-y-6">
      {/* Header */}
      <div>
        <h1 className="text-3xl font-bold text-gray-900">Admin Dashboard</h1>
        <p className="text-gray-600 mt-1">Overview of your student accommodation platform</p>
      </div>

      {/* Stats Grid */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
        {statCards.map((card, index) => (
          <div key={index} className="bg-white rounded-lg shadow-sm p-6">
            <div className="flex items-center justify-between">
              <div>
                <p className="text-sm font-medium text-gray-600">{card.title}</p>
                <p className="text-3xl font-bold text-gray-900 mt-2">{card.value.toLocaleString
                <p className="text-sm text-gray-500 mt-1">{card.change}</p>
              </div>
              <div className={`p-3 rounded-full ${card.color}`}>
                <card.icon className="w-6 h-6 text-white" />
              </div>
```

```
            </div>
          </div>
        ))}
      </div>

      {/* Recent Activity */}
      <div className="bg-white rounded-lg shadow-sm">
        <div className="px-6 py-4 border-b border-gray-200">
          <h2 className="text-lg font-semibold text-gray-900">Recent Admin Activity</h2>
        </div>
        <div className="p-6">
          {recentActivity.length > 0 ? (
            <div className="space-y-4">
              {recentActivity.map((activity, index) => (
                <div key={index} className="flex items-center space-x-4 py-3 border-b border-gr
                  <div className="flex-shrink-0">
                    {activity.action === 'ban_user' && <AlertCircle className="w-5 h-5 text-red
                    {activity.action === 'approve_property' && <CheckCircle className="w-5 h-5
                    {activity.action === 'admin_login' && <Users className="w-5 h-5 text-blue-5
                    {!['ban_user', 'approve_property', 'admin_login'].includes(activity.action)
                      <TrendingUp className="w-5 h-5 text-gray-500" />
                    )}
                  </div>
                  <div className="flex-1">
                    <p className="text-sm font-medium text-gray-900">
                      {activity.admin_name} {activity.action.replace('_', ' ')}
                    </p>
                    <p className="text-sm text-gray-500">
                      {activity.details} • {new Date(activity.created_at).toLocaleString()}
                    </p>
                  </div>
                </div>
              ))}
            </div>
          ) : (
            <p className="text-gray-500 text-center py-8">No recent activity</p>
          )}
        </div>
      </div>

      {/* Quick Actions */}
      <div className="bg-white rounded-lg shadow-sm p-6">
        <h2 className="text-lg font-semibold text-gray-900 mb-4">Quick Actions</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
```

```
            <button
              onClick={() => window.location.href = '/admin/properties?status=pending'}
              className="flex items-center justify-between p-4 border border-gray-200 rounded-lg
            >
              <div>
                <p className="font-medium text-gray-900">Review Properties</p>
                <p className="text-sm text-gray-500">{stats.pendingApprovals} pending</p>
              </div>
              <Clock className="w-5 h-5 text-orange-500" />
            </button>

            <button
              onClick={() => window.location.href = '/admin/users'}
              className="flex items-center justify-between p-4 border border-gray-200 rounded-lg
            >
              <div>
                <p className="font-medium text-gray-900">Manage Users</p>
                <p className="text-sm text-gray-500">{stats.totalUsers} total users</p>
              </div>
              <Users className="w-5 h-5 text-blue-500" />
            </button>

            <button
              onClick={() => window.location.href = '/admin/reviews?status=flagged'}
              className="flex items-center justify-between p-4 border border-gray-200 rounded-lg
            >
              <div>
                <p className="font-medium text-gray-900">Review Flags</p>
                <p className="text-sm text-gray-500">{stats.flaggedReviews} flagged reviews</p>
              </div>
              <AlertCircle className="w-5 h-5 text-red-500" />
            </button>
          </div>
        </div>
      </div>
    );
};

export default AdminDashboard;
```

Phase 5: Deployment & Security (Week 4)

Step 11: Admin Dashboard Stats API

```
python
```

```python
# backend/routes/admin_dashboard.py - Create new file

from flask import Blueprint, jsonify
from flask_jwt_extended import jwt_required
from models.user import User
from models.property import Property
from models.review import Review
from models.admin import AdminLog, PropertyApproval
from utils.admin_decorators import admin_required
from datetime import datetime, timedelta

admin_dashboard_bp = Blueprint('admin_dashboard', __name__)

@admin_dashboard_bp.route('/admin/dashboard/stats', methods=['GET'])
@admin_required()
def get_dashboard_stats():
    """Get dashboard statistics"""
    try:
        # Calculate date ranges
        now = datetime.utcnow()
        week_ago = now - timedelta(days=7)

        # Basic counts
        total_users = User.query.count()
        total_properties = Property.query.count()
        total_reviews = Review.query.count()

        # Pending approvals
        pending_approvals = PropertyApproval.query.filter_by(status='pending').count()

        # This week's stats
        new_users_this_week = User.query.filter(User.created_at >= week_ago).count()

        approved_properties_this_week = PropertyApproval.query.filter(
            PropertyApproval.status == 'approved',
            PropertyApproval.reviewed_at >= week_ago
        ).count()

        # Flagged reviews (you'll need to add a flagged field to Review model)
        flagged_reviews = Review.query.filter_by(flagged=True).count() if hasattr(Review, 'flag

        # Active users (logged in this week)
        active_users = User.query.filter(User.last_login >= week_ago).count()
```

```
        return jsonify({
            'totalUsers': total_users,
            'totalProperties': total_properties,
            'totalReviews': total_reviews,
            'pendingApprovals': pending_approvals,
            'newUsersThisWeek': new_users_this_week,
            'approvedPropertiesThisWeek': approved_properties_this_week,
            'flaggedReviews': flagged_reviews,
            'activeUsers': active_users
        }), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch dashboard stats'}), 500

@admin_dashboard_bp.route('/admin/dashboard/recent-activity', methods=['GET'])
@admin_required()
def get_recent_activity():
    """Get recent admin activity for dashboard"""
    try:
        # Get last 10 admin actions
        recent_logs = AdminLog.query.order_by(
            AdminLog.created_at.desc()
        ).limit(10).all()

        activities = []
        for log in recent_logs:
            activity = {
                'id': log.id,
                'action': log.action,
                'admin_name': log.admin.name,
                'target_type': log.target_type,
                'target_id': log.target_id,
                'created_at': log.created_at.isoformat(),
                'details': format_activity_details(log)
            }
            activities.append(activity)

        return jsonify({
            'activities': activities
        }), 200

    except Exception as e:
        return jsonify({'error': 'Failed to fetch recent activity'}), 500
```

```python
def format_activity_details(log):
    """Format activity details for display"""
    if log.action == 'ban_user':
        return f"Banned user for: {log.details.get('reason', 'No reason provided')}"
    elif log.action == 'approve_property':
        return f"Approved property: {log.details.get('property_name', 'Unknown')}"
    elif log.action == 'admin_login':
        return f"Logged in from {log.details.get('ip_address', 'Unknown IP')}"
    elif log.action == 'view_users':
        return f"Viewed users list (page {log.details.get('page', 1)})"
    else:
        return f"Performed {log.action} on {log.target_type} #{log.target_id}"
```

Step 12: Register All Admin Routes

python

```python
# backend/app/__init__.py - Update your app initialization

from flask import Flask
from routes.admin_auth import admin_auth_bp
from routes.admin_users import admin_users_bp
from routes.admin_properties import admin_properties_bp
from routes.admin_dashboard import admin_dashboard_bp

def create_app():
    app = Flask(__name__)

    # Your existing configuration...

    # Register admin blueprints
    app.register_blueprint(admin_auth_bp, url_prefix='/api')
    app.register_blueprint(admin_users_bp, url_prefix='/api')
    app.register_blueprint(admin_properties_bp, url_prefix='/api')
    app.register_blueprint(admin_dashboard_bp, url_prefix='/api')

    return app
```

Step 13: Frontend Admin Routes

jsx

```jsx
// frontend/src/App.js - Add admin routes

import AdminRoute from './components/AdminRoute';
import AdminLayout from './components/Admin/AdminLayout';
import AdminDashboard from './pages/Admin/AdminDashboard';
import AdminUsers from './pages/Admin/AdminUsers';
import AdminProperties from './pages/Admin/AdminProperties';
import AdminLogin from './pages/Admin/AdminLogin';

function App() {
  return (
    <Router>
      <Routes>
        {/* Regular routes */}
        <Route path="/" element={<HomePage />} />
        <Route path="/properties" element={<PropertiesPage />} />
        <Route path="/reviews" element={<ReviewsPage />} />

        {/* Admin routes */}
        <Route path="/admin/login" element={<AdminLogin />} />
        <Route path="/admin/*" element={
          <AdminRoute>
            <AdminLayout>
              <Routes>
                <Route path="/" element={<AdminDashboard />} />
                <Route path="/users" element={
                  <AdminRoute requiredPermission="manage_users">
                    <AdminUsers />
                  </AdminRoute>
                } />
                <Route path="/properties" element={
                  <AdminRoute requiredPermission="approve_properties">
                    <AdminProperties />
                  </AdminRoute>
                } />
              </Routes>
            </AdminLayout>
          </AdminRoute>
        } />
      </Routes>
    </Router>
```

```
  );
}
```

🎯 Implementation Timeline & Checklist

Week 1: Foundation ✅

- [ ] Update User model with admin fields
- [ ] Create admin-specific models (AdminLog, PropertyApproval)
- [ ] Run database migrations
- [ ] Create first admin user script
- [ ] Set up admin authentication middleware

Week 2: Core APIs ✅

- [ ] Implement admin authentication routes
- [ ] Build user management APIs
- [ ] Create property approval APIs
- [ ] Add admin activity logging
- [ ] Test all APIs with Postman

Week 3: Frontend Interface ✅

- [ ] Create admin route protection
- [ ] Build admin layout component
- [ ] Implement admin dashboard
- [ ] Create user management interface
- [ ] Build property approval interface

Week 4: Polish & Security ✅

- [ ] Add comprehensive error handling
- [ ] Implement proper input validation
- [ ] Add admin activity monitoring
- [ ] Create admin documentation
- [ ] Deploy and test production setup

🔒 Security Best Practices

Input Validation

python

```python
# Example validation for admin actions
def validate_admin_input(data, required_fields):
    errors = []

    for field in required_fields:
        if field not in data or not data[field]:
            errors.append(f'{field} is required')

    if errors:
        return {'valid': False, 'errors': errors}

    return {'valid': True, 'errors': []}
```

Rate Limiting

python

```python
# Add to admin routes for additional security
from flask_limiter import Limiter

limiter = Limiter(
    app,
    key_func=lambda: get_jwt_identity(),
    default_limits=["1000 per hour"]
)

@admin_users_bp.route('/admin/users/<int:user_id>/ban', methods=['POST'])
@limiter.limit("10 per minute")  # Limit ban actions
@admin_required('manage_users')
def ban_user(user_id):
    # ... implementation
```

Audit Trail

python

```python
# Comprehensive logging for all admin actions
def log_detailed_admin_action(action, target_type, target_id, details=None, sensitive=False):
    log_entry = AdminLog(
        admin_id=request.current_admin.id,
        action=action,
        target_type=target_type,
        target_id=target_id,
        details=details if not sensitive else {'action': 'sensitive_operation'},
        ip_address=request.remote_addr,
        user_agent=request.headers.get('User-Agent', ''),
        created_at=datetime.utcnow()
    )

    db.session.add(log_entry)
    db.session.commit()

    # Optional: Send to external logging service for production
    if app.config.get('EXTERNAL_LOGGING'):
        send_to_external_logger(log_entry)
```

This comprehensive guide provides everything you need to implement a full-featured admin site for your student accommodation platform! 🚀