Smart Classroom Booking System - Complete Code Documentation

Table of Contents

System Overview

The Smart Classroom Booking System is a Django-based web application designed to manage bookings for a single smart classroom. The system provides:

- **User Management**: Registration, authentication, and profile management
- **Booking System**: Time slot booking with conflict resolution
- **Admin Panel**: Customized Django admin for booking approval/rejection
- **Email Notifications**: Automated emails for confirmations and status updates
- **Calendar Interface**: Visual weekly calendar for booking management
- **Receipt Management**: File upload and storage for payment receipts

Technology Stack

- **Backend**: Django 4.2.23
- **Database**: SQL Server with ODBC Driver
- **Frontend**: HTML, CSS (Tailwind), JavaScript
- **Email**: Gmail SMTP
- **File Storage**: Local file system

Project Structure

```
Smart Classroom Booking System/
├── Admin/                          # Main Django project directory
│   ├── manage.py                   # Django management script
│   ├── db.sqlite3                  # SQLite database file
│   ├── Admin/                      # Project configuration
│   │   ├── settings.py             # Django settings
│   │   ├── urls.py                 # Main URL configuration
│   │   ├── views.py                # Main project views
│   │   └── wsgi.py                 # WSGI configuration
│   ├── accounts/                   # User authentication app
│   │   ├── forms.py                # Custom user registration forms
│   │   └── backends.py             # Email authentication backend
│   ├── Booking/                    # Core booking functionality
│   │   ├── models.py               # Database models
│   │   ├── views.py                # Booking views and logic
│   │   ├── forms.py                # Booking forms
│   │   ├── admin.py                # Admin interface customization
│   │   ├── email_utils.py          # Email functionality
│   │   ├── profile_models.py       # User profile models
│   │   └── templates/              # Booking-specific templates
│   ├── templates/                  # Global templates
│   ├── static/                     # Static files
│   └── media/                      # User uploads
└── todo.md                         # Development checklist
```

Django Configuration

settings.py Analysis

Database Configuration

```
DATABASES = {
    'default': {
        'ENGINE': 'mssql',
        'NAME': 'SmartBookingSystem',
        'HOST': r'GEORGEACER\SQLEXPRESS',
        'OPTIONS': {
            'driver': 'ODBC Driver 17 for SQL Server',
            'trusted_connection': 'yes',
        },
    }
}
```

Purpose: Configures SQL Server connection using Windows authentication for production-level database management.

Email Configuration

```
 EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'smartbookingusp@gmail.com'
EMAIL_HOST_PASSWORD = 'your_password'
DEFAULT_FROM_EMAIL = 'Smart Booking System <smartbookingusp@gmail.com>'
```

Purpose: Enables automated email notifications using Gmail SMTP.

Media and Static Files

```
 MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
STATIC_ROOT = BASE_DIR / 'staticfiles'
```

Purpose: Configures file upload handling and static asset serving.

Installed Apps

```
INSTALLED_APPS = [
    'admin_interface',          # Enhanced admin UI
    'colorfield',              # Color picker fields
    'django.contrib.admin',    # Django admin
    'django.contrib.auth',     # Authentication
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'accounts',                # Custom user management
    'Booking',                 # Core booking functionality
]
```

Purpose: Defines all Django applications and third-party packages used in the project.

Models & Database

Booking Model (`models.py`)

```
class Booking(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    start_time = models.DateTimeField()
    end_time = models.DateTimeField()
    purpose = models.CharField(max_length=100)
    attendees = models.PositiveIntegerField()
    description = models.TextField(blank=True)
    receipt = models.ImageField(upload_to='receipts/', blank=True, null=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Field Analysis:

- **user**: Foreign key linking booking to Django User model
- **start_time/end_time**: DateTime fields for booking duration
- **purpose**: Short description of booking reason (max 100 chars)
- **attendees**: Number of people attending (positive integer only)
- **description**: Optional detailed description
- **receipt**: Image upload for payment verification
- **status**: Choice field (pending/approved/rejected)
- **created_at/updated_at**: Automatic timestamp tracking

Status Choices:

```
STATUS_CHOICES = [
    ('pending', 'Pending'),
    ('approved', 'Approved'),
    ('rejected', 'Rejected'),
]
```

Profile Model (`profile_models.py`)

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    profile_picture = models.ImageField(
        upload_to='profile_pics/',
        default='default.jpg',
        blank=True,
        null=True
    )
    phone_number = models.CharField(max_length=15, blank=True)
    department = models.CharField(max_length=100, blank=True)
    student_id = models.CharField(max_length=20, blank=True)
```

Purpose: Extends Django's User model with additional profile information.

Views & Logic

Booking Calendar View (`views.py`)

```
def booking_view(request):
    # Get week offset from URL parameter (default to current week)
    week_offset = int(request.GET.get('week', 0))

    # Calculate the start of the target week
    today = datetime.today()
    current_week_start = today - timedelta(days=today.weekday())
    start = current_week_start + timedelta(weeks=week_offset)

    # Generate day_dates for the target week
    day_dates = [(
        (start + timedelta(days=i)).strftime('%A'),
        (start + timedelta(days=i)).strftime('%Y-%m-%d')
    ) for i in range(7)]

    hours = list(range(7, 23))  # 7AM to 10PM

    # Only show bookings for this specific week
    week_start = start.replace(hour=0, minute=0, second=0, microsecond=0)
    week_end = week_start + timedelta(days=7)
    bookings = Booking.objects.filter(
        start_time__gte=week_start,
        start_time__lt=week_end
    )
```

Key Features:

1. **Week Navigation**: Uses `week_offset` parameter to navigate between weeks
2. **Date Calculation**: Calculates Monday-Sunday week ranges
3. **Time Slots**: Shows 7AM-10PM hourly slots
4. **Filtered Bookings**: Only loads bookings for the current week view

Color Coding System:

```
color_palette = [
    "from-green-400/95 to-green-600/95",
    "from-blue-400/95 to-blue-600/95",
    "from-cyan-400/95 to-cyan-600/95",
    "from-teal-400/95 to-teal-600/95",
    "from-indigo-400/95 to-indigo-600/95",
    "from-pink-400/95 to-pink-600/95",
    "from-yellow-400/95 to-yellow-500/95",
    "from-purple-400/95 to-purple-600/95",
    "from-rose-400/95 to-rose-600/95",
    "from-orange-400/95 to-orange-600/95",
]

# Assign a color to each user
user_ids = list(User.objects.values_list('id', flat=True).order_by('id'))
user_color_map = {}
for idx, user_id in enumerate(user_ids):
    user_color_map[user_id] = color_palette[idx % len(color_palette)]
```

Purpose: Assigns unique Tailwind CSS gradient colors to each user for visual distinction in the calendar.

Create Booking View

```
@login_required
def create_booking(request):
    # Pre-fill form if slot info is passed
    initial = {}
    date = request.GET.get('date')
    hour = request.GET.get('hour')
    if date and hour:
        try:
            booking_date = datetime.strptime(date, "%Y-%m-%d").date()
            initial['booking_date'] = booking_date
            initial['time_slot'] = f"{hour:02d}:00"
        except Exception:
            pass

    if request.method == 'POST':
        form = BookingForm(request.POST, request.FILES)
        if form.is_valid():
            booking = form.save(commit=False)
            booking.user = request.user
            booking.status = 'pending'

            # Handle receipt upload
            if 'receipt' in request.FILES:
                booking.receipt = request.FILES['receipt']

            booking.save()

            # Send confirmation email to user
            if request.user.email:
                email_sent = send_booking_confirmation_email(booking)
                if email_sent:
                    messages.success(request, 'Booking created successfully! A confirmation email has been sent.')
                else:
                    messages.success(request, 'Booking created successfully!')
                    messages.warning(request, 'Confirmation email could not be sent.')

            # Send notification email to admins
            send_admin_notification_email(booking)

            return redirect('booking')
    else:
        form = BookingForm(initial=initial)

    return render(request, 'create_booking.html', {'form': form})
```

Process Flow:

1. **URL Parameters**: Checks for pre-selected date/hour from calendar clicks
2. **Form Processing**: Validates and saves booking data
3. **File Upload**: Handles receipt image upload
4. **Email Notifications**: Sends confirmation to user and notification to admins
5. **Redirect**: Returns to calendar view after successful creation

---

Forms & Validation

BookingForm (`forms.py`)

```
class BookingForm(forms.ModelForm):
    TIME_SLOT_CHOICES = [
        ('08:00', '8:00 AM - 9:00 AM'),
        ('09:00', '9:00 AM - 10:00 AM'),
        ('10:00', '10:00 AM - 11:00 AM'),
        ('11:00', '11:00 AM - 12:00 PM'),
        ('12:00', '12:00 PM - 1:00 PM'),
        ('13:00', '1:00 PM - 2:00 PM'),
        ('14:00', '2:00 PM - 3:00 PM'),
        ('15:00', '3:00 PM - 4:00 PM'),
        ('16:00', '4:00 PM - 5:00 PM'),
        ('17:00', '5:00 PM - 6:00 PM'),
    ]

    booking_date = forms.DateField(
        widget=forms.DateInput(attrs={
            'type': 'date',
            'class': 'w-full px-4 py-3 border border-blue-200 rounded-xl focus:ring-2 focus:ring-[#2EC4B6] focus:border-[#0E6A6A] bg
        })
    )

    time_slot = forms.ChoiceField(
        choices=TIME_SLOT_CHOICES,
        widget=forms.Select(attrs={
            'class': 'w-full px-4 py-3 border border-blue-200 rounded-xl focus:ring-2 focus:ring-[#2EC4B6] focus:border-[#0E6A6A] bg
        })
    )
```

Form Validation Logic:

```
def clean(self):
    cleaned_data = super().clean()
    booking_date = cleaned_data.get('booking_date')
    time_slot = cleaned_data.get('time_slot')

    if booking_date and time_slot:
        # Check if booking is in the past
        start_time = datetime.combine(booking_date, datetime.strptime(time_slot, '%H:%M').time())
        if start_time <= datetime.now():
            raise forms.ValidationError("Cannot book time slots in the past.")

        # Calculate end time (1 hour later)
        end_time = start_time + timedelta(hours=1)

        # Check for existing bookings in the same time slot
        existing_bookings = Booking.objects.filter(
            start_time__lt=end_time,
            end_time__gt=start_time,
            status__in=['pending', 'approved']
        ).exclude(pk=self.instance.pk if self.instance else None)

        if existing_bookings.exists():
            raise forms.ValidationError("This time slot is already booked.")

        # Set the calculated start and end times
        cleaned_data['start_time'] = start_time
        cleaned_data['end_time'] = end_time

    return cleaned_data

def save(self, commit=True):
    instance = super().save(commit=False)
    instance.start_time = self.cleaned_data['start_time']
    instance.end_time = self.cleaned_data['end_time']
    if commit:
        instance.save()
    return instance
```

Validation Rules:

1. **Past Date Check**: Prevents booking time slots in the past
2. **Conflict Detection**: Checks for overlapping bookings
3. **Status Filtering**: Only considers pending/approved bookings for conflicts
4. **Auto-calculation**: Automatically sets end_time as start_time + 1 hour

Custom User Registration Form

```
class CustomUserRegistrationForm(UserCreationForm):
    email = forms.EmailField(
        required=True,
        help_text='Required. Enter a valid email address.',
        widget=forms.EmailInput(attrs={
            'class': 'w-full px-4 py-2 border border-blue-200 rounded-xl focus:ring-2 focus:ring-[#2EC4B6] focus:border-[#0E6A6A] bg
            'placeholder': 'Your email address'
        })
    )

    def clean_email(self):
        email = self.cleaned_data.get('email')
        if User.objects.filter(email=email).exists():
            raise forms.ValidationError("A user with this email already exists.")
        return email
```

Features:

- Email uniqueness validation
- Custom styling with Tailwind CSS
- Integration with Django's UserCreationForm

Templates & UI

Calendar Template (`booking.html`)

Calendar Grid Structure:

```
<table class="min-w-full bg-gradient-to-br from-blue-50 to-cyan-50 rounded-2xl shadow-2xl overflow-hidden border-2 border-blue-100">
    <thead>
        <tr class="bg-gradient-to-r from-[#0E6A6A] to-[#2EC4B6] text-white">
            <th class="py-4 px-6 text-left font-bold text-lg border-r border-blue-300/30">Time</th>
            {% for day_name, day_date in day_dates %}
                <th class="py-4 px-6 text-center font-bold text-lg border-r border-blue-300/30 last:border-r-0">
                    <div class="text-lg">{{ day_name }}</div>
                    <div class="text-sm opacity-90 mt-1">{{ day_date|date:"M d" }}</div>
                </th>
            {% endfor %}
        </tr>
    </thead>
    <tbody>
        {% for hour in hours %}
            <tr class="border-b border-blue-100/50 hover:bg-blue-50/50 transition-colors duration-200">
                <td class="py-3 px-6 font-semibold text-[#0E6A6A] bg-blue-50/30 border-r border-blue-200/50">
                    {{ hour }}:00
                </td>
                {% for day_name, day_date in day_dates %}
                    <td class="relative p-1 border-r border-blue-100/30 last:border-r-0 h-16">
                        {% for booking in bookings %}
                            {% if booking.start_time|date:"Y-m-d" == day_date and booking.start_time.hour == hour %}
                                <div class="booking-slot bg-gradient-to-r {{ user_color_map|dict_get:booking.user.id }} text-white r
                                    <div class="absolute inset-0 bg-white/10 opacity-0 hover:opacity-100 transition-opacity duration
                                    <div class="relative z-10">
                                        <div class="font-bold truncate">{{ booking.user.first_name|default:booking.user.username }}<
                                        <div class="text-white/90 truncate text-xs mt-1">{{ booking.purpose }}</div>
                                        <div class="text-white/80 text-xs">{{ booking.attendees }} people</div>
                                    </div>
                                </div>
                            {% endif %}
                        {% empty %}
                        {% endfor %}

                        {% comment %} Check if slot is empty {% endcomment %}
                        {% with slot_has_booking=False %}
                            {% for booking in bookings %}
                                {% if booking.start_time|date:"Y-m-d" == day_date and booking.start_time.hour == hour %}
                                    {% with slot_has_booking=True %}{% endwith %}
                                {% endif %}
                            {% endfor %}

                            {% if not slot_has_booking %}
                                <a href="{% url 'create_booking' %}?date={{ day_date }}&hour={{ hour }}&week={{ week_offset }}"
                                   class="block w-full h-full rounded-lg border-2 border-dashed border-blue-200/60 hover:border-[#2E
                                    <span class="text-blue-400/60 group-hover:text-[#2EC4B6] text-xs font-medium transition-colors d
                                        + Book
                                    </span>
                                </a>
                            {% endif %}
                        {% endwith %}
                    </td>
                {% endfor %}
            </tr>
        {% endfor %}
    </tbody>
</table>
```

Key UI Features:

1. **Responsive Grid**: Adapts to different screen sizes
2. **Color Coding**: Each user gets a unique gradient color
3. **Hover Effects**: Interactive feedback on hover
4. **Empty Slot Links**: Clickable areas to create new bookings
5. **Booking Details**: Shows user name, purpose, and attendee count

Navigation Controls:

```
<div class="flex items-center justify-between mb-6">
    <a href="?week={{ prev_week }}"
       class="flex items-center space-x-2 px-6 py-3 bg-gradient-to-r from-blue-500 to-blue-600 text-white rounded-xl hover:from-blue
        <i class="fas fa-chevron-left"></i>
        <span>Previous Week</span>
    </a>

    <div class="text-center">
        <h2 class="text-2xl font-bold text-[#0E6A6A]">{{ week_display }}</h2>
        {% if is_current_week %}
            <span class="inline-block px-3 py-1 bg-green-100 text-green-800 rounded-full text-sm font-medium mt-1">
                Current Week
            </span>
        {% endif %}
    </div>

    <a href="?week={{ next_week }}"
       class="flex items-center space-x-2 px-6 py-3 bg-gradient-to-r from-blue-500 to-blue-600 text-white rounded-xl hover:from-blue
        <span>Next Week</span>
        <i class="fas fa-chevron-right"></i>
    </a>
</div>
```

Create Booking Template (`create_booking.html`)

Form Structure:

```
<form method="post" enctype="multipart/form-data" class="space-y-6">
    {% csrf_token %}

    <div class="grid grid-cols-1 md:grid-cols-2 gap-6">
        <div>
            <label for="{{ form.booking_date.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
                ⍰ Booking Date
            </label>
            {{ form.booking_date }}
        </div>

        <div>
            <label for="{{ form.time_slot.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
                ⍰ Time Slot
            </label>
            {{ form.time_slot }}
        </div>
    </div>

    <div>
        <label for="{{ form.purpose.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
            ⍰ Purpose
        </label>
        {{ form.purpose }}
    </div>

    <div>
        <label for="{{ form.attendees.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
            ⍰ Number of Attendees
        </label>
        {{ form.attendees }}
    </div>

    <div>
        <label for="{{ form.description.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
            ⍰ Description (Optional)
        </label>
        {{ form.description }}
    </div>

    <div>
        <label for="{{ form.receipt.id_for_label }}" class="block text-sm font-bold text-[#0E6A6A] mb-2">
            ⍰ Payment Receipt (Required)
        </label>
        <div class="relative">
            {{ form.receipt }}
            <span id="file-name" class="mt-2 text-sm text-gray-600"></span>
        </div>
    </div>
</form>
```

JavaScript Enhancements:

```
function updateFileName(input) {
    const fileName = document.getElementById('file-name');
    if (input.files && input.files[0]) {
        fileName.textContent = `Selected: ${input.files[0].name}`;
        fileName.className = 'mt-2 text-sm text-green-600 font-medium';
    } else {
        fileName.textContent = '';
    }
}

// Auto-attach to file input
document.addEventListener('DOMContentLoaded', function() {
    const fileInput = document.querySelector('input[type="file"]');
    if (fileInput) {
        fileInput.addEventListener('change', function() {
            updateFileName(this);
        });
    }
});
```

Admin Interface

Custom Admin Configuration (`admin.py`)

Booking Admin Class:

```
@admin.register(Booking)
class BookingAdmin(admin.ModelAdmin):
    list_display = ['user', 'booking_date', 'time_range', 'purpose', 'attendees', 'status_badge', 'receipt_link', 'created_at']
    list_filter = ['status', 'start_time', 'created_at', 'user']
    search_fields = ['user__username', 'user__email', 'purpose', 'description']
    readonly_fields = ['created_at', 'updated_at']
    actions = [approve_bookings, reject_bookings]

    fieldsets = (
        ('Booking Information', {
            'fields': ('user', 'start_time', 'end_time', 'purpose', 'attendees', 'description')
        }),
        ('Status & Receipt', {
            'fields': ('status', 'receipt')
        }),
        ('Timestamps', {
            'fields': ('created_at', 'updated_at'),
            'classes': ('collapse',)
        }),
    )

    def booking_date(self, obj):
        return obj.start_time.strftime('%Y-%m-%d') if obj.start_time else '-'
    booking_date.short_description = 'Date'
    booking_date.admin_order_field = 'start_time'

    def time_range(self, obj):
        if obj.start_time and obj.end_time:
            return f"{obj.start_time.strftime('%I:%M %p')} - {obj.end_time.strftime('%I:%M %p')}"
        return '-'
    time_range.short_description = 'Time'

    def status_badge(self, obj):
        colors = {
            'pending': '#fbbf24',  # Yellow
            'approved': '#10b981', # Green
            'rejected': '#ef4444'  # Red
        }
        color = colors.get(obj.status, '#6b7280')
        return format_html(
            '<span style="background-color: {}; color: white; padding: 4px 12px; border-radius: 20px; font-size: 12px; font-weight:
            color, obj.status
        )
    status_badge.short_description = 'Status'

    def receipt_link(self, obj):
        if obj.receipt:
            return format_html(
                '<a href="{}" target="_blank" style="color: #2563eb; text-decoration: none; font-weight: 500;">□ View Receipt</a>',
                obj.receipt.url
            )
        return '□ No Receipt'
    receipt_link.short_description = 'Receipt'
```

Custom Admin Actions:

```
def approve_bookings(modeladmin, request, queryset):
    """Custom admin action to approve bookings and send notification emails"""
    updated = 0
    for booking in queryset.filter(status='pending'):
        booking.status = 'approved'
        booking.save()

        # Send status update email
        try:
            send_booking_status_email(booking)
        except Exception as e:
            messages.warning(request, f'Email notification failed for booking {booking.id}: {str(e)}')

        updated += 1

    if updated:
        messages.success(request, f'Successfully approved {updated} booking(s) and sent notification emails.')
    else:
        messages.info(request, 'No pending bookings were selected.')

def reject_bookings(modeladmin, request, queryset):
    """Custom admin action to reject bookings and send notification emails"""
    updated = 0
    for booking in queryset.filter(status='pending'):
        booking.status = 'rejected'
        booking.save()

        # Send status update email
        try:
            send_booking_status_email(booking)
        except Exception as e:
            messages.warning(request, f'Email notification failed for booking {booking.id}: {str(e)}')

        updated += 1

    if updated:
        messages.success(request, f'Successfully rejected {updated} booking(s) and sent notification emails.')
    else:
        messages.info(request, 'No pending bookings were selected.')
```

Admin Dashboard Customization (`index.html`):

```
<div class="admin-dashboard">
    <div class="welcome-hero">
        <div class="welcome-title">□ Smart Classroom Admin Panel</div>
        <div class="welcome-subtitle">Manage classroom bookings with ease</div>
    </div>

    <div class="dashboard-grid">
        <div class="stats-section">
            <div class="stats-header">
                <div class="stats-icon">□</div>
                <div class="stats-title">System Overview</div>
            </div>

            <div class="stats-grid">
                <div class="stat-item">
                    <div class="stat-number">{{ total_bookings|default:"0" }}</div>
                    <div class="stat-label">Total Bookings</div>
                </div>

                <div class="stat-item pending">
                    <div class="stat-number">{{ pending_bookings|default:"0" }}</div>
                    <div class="stat-label">Pending Approval</div>
                </div>

                <div class="stat-item">
                    <div class="stat-number">{{ total_users|default:"0" }}</div>
                    <div class="stat-label">Registered Users</div>
                </div>

                <div class="stat-item today">
                    <div class="stat-number">{{ todays_bookings|default:"0" }}</div>
                    <div class="stat-label">Today's Bookings</div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Email System

Email Utilities (`email_utils.py`)

Welcome Email Function:

```
def send_welcome_email(user):
    """Send welcome email to newly registered user"""
    print(f"DEBUG: Starting to send welcome email to {user.email}")

    subject = 'Welcome to Smart Classroom Booking System!'

    try:
        html_message = render_to_string('emails/welcome_email.html', {
            'user': user,
            'site_name': 'Smart Classroom Booking System',
            'login_url': 'http://127.0.0.1:8000/login/',
            'contact_email': 'smartbookingusp@gmail.com'
        })
        print("DEBUG: HTML template rendered successfully")
    except Exception as e:
        print(f"DEBUG: Error rendering template: {e}")
        return False

    plain_message = strip_tags(html_message)

    try:
        result = send_mail(
            subject=subject,
            message=plain_message,
            html_message=html_message,
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=[user.email],
            fail_silently=False,
        )
        print(f"DEBUG: Email send result: {result}")
        return True
    except Exception as e:
        print(f"DEBUG: Error sending email: {e}")
        return False
```

Booking Confirmation Email:

```
def send_booking_confirmation_email(booking):
    """Send booking confirmation email to user"""
    if not booking.user.email:
        print("DEBUG: User has no email address")
        return False

    subject = f'Booking Confirmation - {booking.start_time.strftime("%B %d, %Y")}'

    try:
        html_message = render_to_string('emails/booking_confirmation.html', {
            'booking': booking,
            'user': booking.user,
            'site_name': 'Smart Classroom Booking System',
            'booking_date': booking.start_time.strftime('%A, %B %d, %Y'),
            'booking_time': f"{booking.start_time.strftime('%I:%M %p')} - {booking.end_time.strftime('%I:%M %p')}",
        })
    except Exception as e:
        print(f"DEBUG: Error rendering booking confirmation template: {e}")
        return False

    plain_message = strip_tags(html_message)

    try:
        result = send_mail(
            subject=subject,
            message=plain_message,
            html_message=html_message,
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=[booking.user.email],
            fail_silently=False,
        )
        return True
    except Exception as e:
        print(f"DEBUG: Error sending booking confirmation email: {e}")
        return False
```

Admin Notification Email:

```
def send_admin_notification_email(booking):
    """Send notification email to admins about new booking"""
    admin_emails = ['smartbookingusp@gmail.com']  # Add more admin emails as needed

    subject = f'New Booking Request - {booking.start_time.strftime("%B %d, %Y")}'

    try:
        html_message = render_to_string('emails/admin_notification.html', {
            'booking': booking,
            'user': booking.user,
            'booking_date': booking.start_time.strftime('%A, %B %d, %Y'),
            'booking_time': f"{booking.start_time.strftime('%I:%M %p')} - {booking.end_time.strftime('%I:%M %p')}",
            'admin_url': 'http://127.0.0.1:8000/admin/Booking/booking/',
        })
    except Exception as e:
        print(f"DEBUG: Error rendering admin notification template: {e}")
        return False

    plain_message = strip_tags(html_message)

    try:
        result = send_mail(
            subject=subject,
            message=plain_message,
            html_message=html_message,
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=admin_emails,
            fail_silently=False,
        )
        return True
    except Exception as e:
        print(f"DEBUG: Error sending admin notification email: {e}")
        return False
```

Email Templates

Welcome Email Template (`welcome_email.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome to {{ site_name }}</title>
    <style>
        body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; }
        .container { max-width: 600px; margin: 0 auto; padding: 20px; }
        .header {
            background: linear-gradient(135deg, #0E6A6A, #2EC4B6);
            color: white;
            padding: 30px;
            text-align: center;
            border-radius: 10px 10px 0 0;
        }
        .content {
            background: #f9f9f9;
            padding: 30px;
            border-radius: 0 0 10px 10px;
        }
        .button {
            display: inline-block;
            background: linear-gradient(135deg, #0E6A6A, #2EC4B6);
            color: white;
            padding: 12px 24px;
            text-decoration: none;
            border-radius: 5px;
            margin: 20px 0;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1> Welcome to {{ site_name }}!</h1>
            <p>Your account has been successfully created</p>
        </div>

        <div class="content">
            <h2>Hello {{ user.first_name|default:user.username }}!</h2>

            <p>Thank you for registering with the Smart Classroom Booking System. You can now:</p>

            <ul>
                <li> Book time slots for the smart classroom</li>
                <li> View your booking history</li>
                <li>✉ Receive email notifications about your bookings</li>
                <li> Manage your profile</li>
            </ul>

            <p>Ready to get started?</p>
            <a href="{{ login_url }}" class="button">Login to Your Account</a>

            <hr style="margin: 30px 0; border: none; border-top: 1px solid #ddd;">

            <p><strong>Need help?</strong> Contact us at <a href="mailto:{{ contact_email }}">{{ contact_email }}</a></p>

            <p style="color: #666; font-size: 14px;">
                This is an automated message from the Smart Classroom Booking System.
                Please do not reply to this email.
            </p>
        </div>
    </div>
</body>
</html>
```

Authentication

Custom Authentication Backend (`backends.py`)

```
class EmailBackend(ModelBackend):
    """
    Authenticate using email address.
    """
    def authenticate(self, request, username=None, password=None, **kwargs):
        try:
            # Try to find user by email or username
            user = User.objects.get(
                Q(username=username) | Q(email=username)
            )
        except User.DoesNotExist:
            return None

        if user.check_password(password) and self.user_can_authenticate(user):
            return user
        return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

Purpose: Allows users to login using either username or email address.

Registration View (`views.py`):

```
@csrf_protect
def register_view(request):
    if request.method == 'POST':
        form = CustomUserRegistrationForm(request.POST)
        if form.is_valid():
            user = form.save()

            # Create user profile
            Profile.objects.get_or_create(user=user)

            # Send welcome email
            if user.email:
                try:
                    send_welcome_email(user)
                    messages.success(request, 'Registration successful! Welcome email sent.')
                except Exception as e:
                    messages.success(request, 'Registration successful!')
                    messages.warning(request, 'Welcome email could not be sent.')

            # Auto-login the user
            login(request, user)
            messages.success(request, f'Welcome {user.first_name or user.username}!')
            return redirect('dashboard')
    else:
        form = CustomUserRegistrationForm()

    return render(request, 'registration/register.html', {'form': form})
```

Static Files & Styling

Custom Admin CSS (`custom_admin.css`)

Color Variables:

```
:root {
    --primary-color: #0E6A6A;
    --secondary-color: #2EC4B6;
    --accent-color: #FFE66D;
    --success-color: #10b981;
    --warning-color: #f59e0b;
    --danger-color: #ef4444;
    --dark-bg: #1f2937;
    --light-bg: #f8fafc;
}
```

Header Styling:

```
#header {
    background: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
    border-bottom: 3px solid var(--accent-color);
    box-shadow: 0 4px 20px rgba(14, 106, 106, 0.3);
    position: relative;
    overflow: hidden;
}

#header::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: url('data:image/svg+xml,<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100"><defs><pattern id="grain" widt
    pointer-events: none;
}
```

Animation Effects:

```
@keyframes bounce {
    0%, 20%, 50%, 80%, 100% { transform: translateY(0); }
    40% { transform: translateY(-10px); }
    60% { transform: translateY(-5px); }
}

@keyframes shimmer {
    0% { background-position: -200px 0; }
    100% { background-position: 200px 0; }
}

.module h2::after {
    content: '';
    position: absolute;
    top: 0;
    left: -100%;
    width: 100%;
    height: 100%;
    background: linear-gradient(90deg, transparent, rgba(255, 255, 255, 0.2), transparent);
    animation: shimmer 2s infinite;
}
```

URL Configuration

Main Project URLs (`Admin/urls.py`):

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home_view, name='home'),
    path('dashboard/', views.DashboardView, name='dashboard'),
    path('register/', views.register_view, name='register'),
    path('profile/', views.profile_view, name='profile'),
    path('profile/edit/', views.edit_profile_view, name='edit_profile'),
    path('booking/', include('Booking.urls')),
    path('accounts/', include('django.contrib.auth.urls')),  # Login, logout, etc.
]

# Serve media files during development
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Booking App URLs (`Booking/urls.py`):

```
from django.urls import path
from . import views

urlpatterns = [
    path('book/', views.booking_view, name='booking'),
    path('create/', views.create_booking, name='create_booking'),
]
```

Code Flow Analysis

Complete User Journey

1. Registration Process:

```
User visits /register/
→ CustomUserRegistrationForm displayed
→ Form validation (email uniqueness, password strength)
→ User object created
→ Profile object created automatically
→ Welcome email sent
→ User auto-logged in
→ Redirected to dashboard
```

2. Booking Creation Process:

```
User visits /booking/book/
→ Calendar view with week navigation
→ User clicks empty time slot
→ Redirected to /booking/create/ with pre-filled date/hour
→ BookingForm displayed with validation
→ File upload for receipt processed
→ Conflict checking performed
→ Booking saved with 'pending' status
→ Confirmation email sent to user
→ Admin notification email sent
→ User redirected back to calendar
```

3. Admin Approval Process:

```
Admin receives notification email
→ Admin logs into /admin/
→ Views booking in admin panel
→ Uses bulk action to approve/reject
→ Status updated in database
→ Status notification email sent to user
→ Admin sees success message
```

Database Queries Optimization

Calendar View Optimization:

```python
# Only load bookings for the specific week
week_start = start.replace(hour=0, minute=0, second=0, microsecond=0)
week_end = week_start + timedelta(days=7)
bookings = Booking.objects.filter(
    start_time__gte=week_start,
    start_time__lt=week_end
).select_related('user')  # Optimize user data loading
```

Color Mapping Optimization:

```python
# Load all user IDs once and cache color mapping
user_ids = list(User.objects.values_list('id', flat=True).order_by('id'))
user_color_map = {
    user_id: color_palette[idx % len(color_palette)]
    for idx, user_id in enumerate(user_ids)
}
```

Security Considerations

1. CSRF Protection:

- All forms include `{% csrf_token %}`
- Views use `@csrf_protect` decorator where needed

2. Authentication:

- `@login_required` decorator on sensitive views
- User ownership validation in booking operations

3. File Upload Security:

- Receipt uploads restricted to images
- Files stored in controlled media directory

4. SQL Injection Prevention:

- Django ORM used exclusively
- No raw SQL queries

5. XSS Prevention:

- Template auto-escaping enabled
- `format_html()` used for safe HTML generation

Performance Considerations

1. Database Optimization:

- `select_related('user')` to prevent N+1 queries
- Week-based filtering to limit data loads
- Indexed fields for common queries

2. Static File Handling:

- Separate static and media file serving
- CSS/JS minification ready for production

3. Email Performance:

- Asynchronous email sending recommended for production
- Email template caching

Deployment Considerations

Production Settings Checklist:

1. **Database Configuration**:
    - Migrate from SQLite to SQL Server
    - Configure connection pooling
    - Set up database backups
2. **Security Settings**:
    - Set `DEBUG = False`
    - Configure `ALLOWED_HOSTS`
    - Use environment variables for secrets
3. **Email Configuration**:
    - Configure production SMTP settings
    - Set up email monitoring
4. **Static Files**:
    - Configure `STATIC_ROOT` for collection
    - Set up CDN for media files
5. **Monitoring**:
    - Add logging configuration
    - Set up error tracking
    - Configure performance monitoring

Future Enhancement Opportunities

1. Advanced Features:

- Multiple classroom support
- Recurring bookings
- Booking conflicts resolution
- Integration with calendar systems

2. Performance Improvements:

- Redis caching for calendar data
- Database query optimization
- Asynchronous task processing

3. User Experience:

- Real-time booking updates
- Mobile app development
- Push notifications

4. Administrative Features:

- Advanced reporting and analytics
- Automated booking policies
- Integration with payment systems

Conclusion

The Smart Classroom Booking System demonstrates a well-structured Django application with:

- **Clean Architecture**: Separation of concerns with proper app structure
- **User-Friendly Interface**: Modern, responsive design with interactive elements
- **Robust Validation**: Comprehensive form validation and conflict prevention
- **Professional Admin Panel**: Customized Django admin with enhanced functionality
- **Automated Communications**: Email notification system for all stakeholders
- **Security Best Practices**: CSRF protection, authentication, and secure file handling

The codebase is maintainable, scalable, and ready for production deployment with appropriate configuration changes.