TEMPLE DARSHAN TICKET BOOKING

Project submitted to the

APSSDC

Bachelor of Technology In

Computer Science and Engineering

Aditya College of Engineering and Technology

Submitted By

POLIMERU SAI KIRAN-22MH1A0552

INGUVA SIVA RAJANNA PADALA-22MH1A0520

THUSTI NIKHIL-22MH1A0574

SANAPATHI RAKESH-22MH1A0561

PEDHAPUDI SHAKTI VAMSI-22MH1A0547

Under the guidance of

T.SRIKANYA

Y.REVATHI

July 2025

TABLE OF CONTENT

ABSTRACT

The **Temple Darshan Ticket Booking System** is a web-based application developed using the Django web framework. The system is designed to facilitate the online booking of darshan (temple visits) tickets for devotees, ensuring a smooth, efficient, and contactless process for managing temple visits. This solution aims to reduce long queues, manage crowd control, and provide a seamless experience for both temple administrators and devotees.

The application provides essential features such as user registration and login, viewing available darshan slots, real-time slot availability tracking, ticket booking, and payment integration (optional). Admin users can manage slot timings, view bookings, and generate reports for attendance and scheduling. The system uses Django's built-in user authentication, relational database integration (e.g., SQLite/PostgreSQL), and secure session handling to ensure data integrity and user privacy.

This project is especially relevant in the post-pandemic era, where crowd management and time-slot-based scheduling are crucial for maintaining public health and safety. The Temple Darshan Ticket Booking System not only streamlines the booking process but also enhances the overall experience of visiting religious sites.

This system enables devotees to register and log in to a secure platform where they can book tickets for darshan at their preferred time slots. Temple administrators have the ability to manage available slots, view booking data, and control entry flow to ensure a better and safer experience for visitors. The application promotes digital adoption in religious institutions and supports government health and safety guidelines by enabling contactless booking and attendance tracking.

The use of Django ensures rapid development, robust security, and maintainability. This system serves as a useful model for temples and religious organizations looking to adopt digital solutions for crowd control and user convenience.

This paper presents the implementation of a *Temple Darshan Ticket Booking System* using Django, aimed at automating the scheduling and management of temple visits. The system is designed to mitigate the issues of overcrowding, manual ticketing, and inefficient crowd control commonly faced in traditional temple environments. The application allows users to register and book darshan slots online, while administrators manage slot availability and booking limits.

The backend leverages Django ORM for database interaction, along with secure authentication and session management. The system architecture promotes scalability and adaptability for different temple sizes and user volumes. The solution demonstrates the role of modern web technologies in enhancing religious tourism management and offers a robust model for similar crowd-based applications in public spaces.

INTRODUCTION

In recent years, the integration of digital technologies into religious and cultural institutions has significantly improved the way services are delivered to the public. One such area of innovation is the management of temple visits, especially in regions where religious tourism and spiritual gatherings attract large crowds. Traditionally, devotees have had to wait in long queues for darshan (a sacred temple visit or audience with the deity), leading to overcrowding, delays, and often uncomfortable or unsafe conditions, particularly during festivals and peak seasons.

The **Temple Darshan Ticket Booking System** is a web-based application developed using the **Django** web framework, designed to address these challenges by offering a structured, user-friendly, and contactless way for devotees to book darshan tickets online. By allowing users to pre-book time slots for temple visits, the system improves operational efficiency, reduces crowd congestion, and ensures a more organized and peaceful experience for both temple authorities and visitors.

The application consists of two main user roles: **devotees (users)** and **administrators (temple staff)**. Devotees can create an account, view available darshan slots, and reserve tickets for a specific date and time. Administrators can configure slot capacities, manage bookings, monitor attendance, and access reports for decision-making. The system also incorporates secure user authentication, session handling, and a clean interface built with HTML and CSS templates.

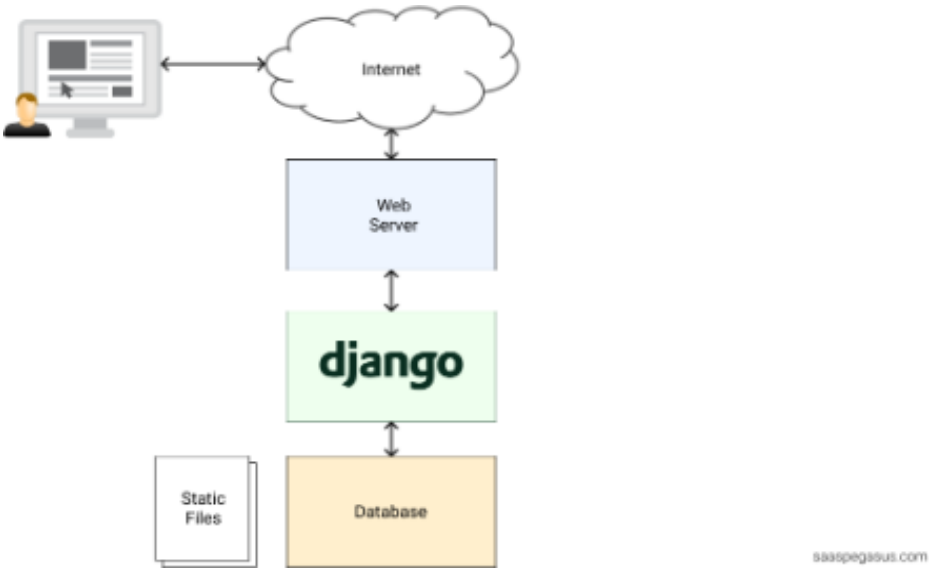
Django, as a high-level Python web framework, provides rapid development and robust security features out-of-the-box, including ORM for database management, built-in authentication systems, and modular structure for scalability. The system uses a relational database (such as SQLite or PostgreSQL) to store user data, slot timings, and booking records.

The primary objective of this project is to leverage technology to simplify and digitize the temple darshan process while ensuring a safe, efficient, and spiritually fulfilling experience for all visitors. This system is particularly relevant in the post-pandemic context, where crowd control and hygiene are of paramount importance. It also sets the foundation for integrating additional services such as donation tracking, VIP access, and QR-code-based entry systems in future enhancements.

SYSTEM REQUIREMENTS

Component	Specification
Operating System	Windows 10/11, Ubuntu 20.04+, or macOS
Backend Framework	Django 4.x (Python Web Framework)
Programming Language	Python 3.10 or above
Database	SQLite (for development) or PostgreSQL/MySQL (for production)
Frontend Technologies	HTML5, CSS3, Bootstrap (optional for styling)
Web Server	Django development server (for local testing) or Gunicorn/Nginx (for deployment)
Browser Compatibility	Chrome, Firefox, Edge, Safari (latest versions)
Package Manager	pip (Python Package Installer)
Other Libraries/Tools	Django Admin, Django ORM, Virtualenv
IDE/Code Editor	VS Code, PyCharm, Sublime Text, or any text editor
Version Control	Git (with GitHub or GitLab for source code hosting)

ARCHITECTURE



The Temple Ticket Booking System is a web application built using the Django framework, which follows the Model-View-Template (MVT) pattern. Users can register by entering basic details and then log in securely. Django checks their login details with the database and either grants access or shows an error if the information is wrong.

After logging in, users can see a list of temples with details like name, location, images, and ticket options (such as General or Special Darshan). They can select a ticket, choose a date and time, and book it. Booking data is stored in the database using Django models.

Once a ticket is booked, a confirmation page shows the booking ID and temple details. Users can also view or cancel their bookings under the "My Bookings" section. Pages like About Us, Contact, FAQs, and Donations are also available, built using Django templates.

Admins have their own login to manage temple information. Through the Django admin panel or a custom dashboard, they can add temples, set ticket types, and view bookings. All data is safely handled using Django's ORM (Object-Relational Mapper).

Overall, this system makes temple bookings easy, organized, and efficient for both users and administrators.

ADVANTAGES

1. User Convenience

- Devotees can book darshan tickets online from anywhere, anytime.
- Eliminates the need to wait in long queues, especially during peak times or festivals.
- Offers flexibility to select preferred date and time slots for darshan.

2. Efficient Crowd Management

- Time-slot-based booking prevents overcrowding and ensures orderly temple visits.
- Reduces stress on temple staff and improves the visitor flow within the temple premises.
- Helps comply with social distancing and public safety norms.

3. Admin Control and Automation

- Admin dashboard for managing darshan slots, bookings, and viewing attendance data.
- Automates ticket generation, reducing manual paperwork and errors.
- Enables reporting and data analysis for better decision-making.

4. Secure and Scalable

- Built using Django's secure authentication and session management features.
- Database-backed system ensures data integrity and can scale for large temples or multiple branches.
- Role-based access ensures only authorized users can manage sensitive operations.

5. Customizable and Expandable

- Can be enhanced with new features such as donation tracking, SMS/email alerts, or QR-code scanning.
- Supports integration with payment gateways for premium darshan or special events.
- The modular design allows easy adaptation for other use cases (e.g., event booking, pilgrimages).

6. Post-COVID Relevance

- Offers a contactless, safe experience for both devotees and temple staff.
- Promotes digital transformation in religious institutions.
- Helps comply with health guidelines and visitor tracking protocols.

7. Eco-Friendly and Paperless

- The system reduces the need for printed tickets and paper records, promoting a more sustainable and eco-friendly approach.
- Digital records make it easier to store and retrieve information without the risk of physical damage or loss.

8. Real-Time Slot Availability

- Users can see real-time availability of darshan slots, reducing booking conflicts.
- Enhances transparency and improves trust in the system, as users know exactly what's available.

PROJECT CODE

```
<!-- templates/temple_app/home.html -->
{% extends 'base.html' %}
{% load static %}

{% block title %}Welcome to Temple Tickets{% endblock %}

{% block content %}
    <div class="hero-section">
        <div class="hero-content">
            <h1>Embark on a Spiritual Journey</h1>
            <p>Discover divine temples and book your visit with ease.</p>
            {# Link the hero button to the featured temples section ID #}
            <a href="#featured-temples-section" class="btn hero-btn">Explore Temples</a>
        </div>
    </div>

    <section class="section other-gods-section">
        <h2>Embrace the Divine Presence</h2>
        <div class="god-images-grid">
            <div class="god-image-item">
                <img src="{% static 'images/lordshiva.jpg' %}" alt="Lord Shiva">
                <p>Lord Shiva</p>
            </div>
            <div class="god-image-item">
                <img src="{% static 'images/goddessparvati.jpg' %}" alt="Goddess Parvati">
                <p>Goddess Parvati</p>
            </div>
            <div class="god-image-item">
                <img src="{% static 'images/lordvishnu.jpg' %}" alt="Lord Vishnu">
                <p>Lord Vishnu</p>
            </div>
            <div class="god-image-item">
                <img src="{% static 'images/goddesslakshmi.jpg' %}" alt="Goddess Lakshmi">
                <p>Goddess lakshmi</p>
            </div>
        </div>
```

```
    </div>
</section>

{# Add an ID to the section containing the temple list #}
<section class="section featured-temples-section" id="featured-temples-section">
    <h2>Featured Temples</h2>
    {% if temples %}
        <div class="temple-list">
            {% for temple in temples %}
                <div class="temple-card">
                    {% if temple.image %}
                        <img src="{{ temple.image.url }}" alt="{{ temple.name }}">
                    {% else %}
                        <img src="{% static 'images/default_temple.jpg' %}" alt="Default Temple Image">
                    {% endif %}
                    <h3><a href="{% url 'temple_detail' temple.pk %}">{{ temple.name }}</a></h3>
                    <p>{{ temple.location }}</p>
                    <p>{{ temple.description|truncatechars:100 }}</p>
                    <a href="{% url 'temple_detail' temple.pk %}" class="btn">View Details</a>
                </div>
            {% endfor %}
        </div>
        <div class="text-center" style="margin-top: 30px;">
            {# Link the "View All Temples" button to the same section ID #}
            <a href="#featured-temples-section" class="btn">View All Temples</a>
        </div>
    {% else %}
        <p style="text-align: center;">No temples listed yet. Please add some through the <a href="{% url 'admin:index' %}">admin panel</a>.</p>
    {% endif %}
</section>
{% endblock %}
```

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

urlpatterns = [
    path('', views.home_view, name='home'),
    path('temples/', views.temple_list_view, name='temple_list'),
    path('temple/<int:pk>/', views.temple_detail_view, name='temple_detail'),
    path('book/<int:ticket_type_pk>/', views.create_booking, name='create_booking'),
    path('booking_success/<int:booking_pk>/', views.booking_success_view, name='booking_success'),
    path('my_bookings/', views.my_bookings_view, name='my_bookings'),
    path('add_temple/', views.add_temple_view, name='add_temple'),
    path('cancel_booking/<int:booking_pk>/', views.cancel_booking_view, name='cancel_booking'),
    path('register/', views.RegisterView.as_view(), name='register'),
    path('about/', views.about_us_view, name='about_us'),
    path('contact/', views.contact_us_view, name='contact_us'),
    path('faq/', views.faq_view, name='faq'),
    path('donate/', views.donations_view, name='donations'),
    path('api/donate/', views.create_donation_ajax, name='api_create_donation'),

    # THIS IS THE LINE THAT NEEDS TO BE MODIFIED:
    # Change 'views.get_donations_data' to 'views.api_donations_data'
    path('api/donations-data/', views.api_donations_data, name='api_donations_data'),
]
```

```python
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.db import transaction
from django.urls import reverse_lazy
from django.views.generic.edit import CreateView
from django.core.mail import send_mail
from django.conf import settings
from django.http import JsonResponse
from django.db.models import Sum
from django.views.decorators.http import require_POST
from django.forms.models import model_to_dict # To easily convert model instances to dicts
import json # To parse JSON for form errors

# Import all your models
from .models import Temple, TicketType, Booking, Donation

# Import all your forms
from .forms import BookingForm, ContactForm, CustomUserCreationForm, TempleForm, DonationForm


# --- Core Temple & Booking Views ---

Tabnine | Edit | Test | Explain | Document
def home_view(request):
    """Displays the home page with a list of temples."""
    temples = Temple.objects.all()
    return render(request, 'temple_app/home.html', {'temples': temples})

Tabnine | Edit | Test | Explain | Document
def temple_list_view(request):
    """Displays a comprehensive list of all temples."""
    temples = Temple.objects.all()
    return render(request, 'temple_app/temple_list.html', {'temples': temples})
```

```python
Tabnine | Edit | Test | Explain | Document
def temple_detail_view(request, pk):
    """Displays details of a specific temple and its ticket types."""
    temple = get_object_or_404(Temple, pk=pk)
    ticket_types = temple.ticket_types.all()
    return render(request, 'temple_app/temple_detail.html', {'temple': temple, 'ticket_types': ticket_types})

Tabnine | Edit | Test | Explain | Document
@login_required
def create_booking(request, ticket_type_pk):
    """Handles the creation of a new booking for a specific ticket type."""
    ticket_type = get_object_or_404(TicketType, pk=ticket_type_pk)

    if request.method == 'POST':
        form = BookingForm(request.POST, ticket_type=ticket_type)
        if form.is_valid():
            quantity = form.cleaned_data['quantity']
            # visit_date = form.cleaned_data['visit_date'] # Already handled by form.save()

            with transaction.atomic():
                ticket_type_for_update = TicketType.objects.select_for_update().get(pk=ticket_type_pk)

                if quantity > ticket_type_for_update.available_tickets:
                    messages.error(request, f"Sorry, only {ticket_type_for_update.available_tickets} tickets are now available for {ticket_type_fo
                    return render(request, 'temple_app/create_booking.html', {'form': form, 'ticket_type': ticket_type_for_update})

                booking = form.save(commit=False)
                booking.user = request.user
                booking.ticket_type = ticket_type_for_update
                booking.status = 'confirmed'

                ticket_type_for_update.available_tickets -= quantity
                ticket_type_for_update.save()

                booking.save()
```

```
                booking.save()
                messages.success(request, 'Your booking has been confirmed!')
                return redirect('booking_success', booking_pk=booking.pk)
            else:
                messages.error(request, 'Please correct the errors in the form.')
        else:
            form = BookingForm(ticket_type=ticket_type)

        return render(request, 'temple_app/create_booking.html', {'form': form, 'ticket_type': ticket_type})

Tabnine | Edit | Test | Explain | Document
@login_required
def booking_success_view(request, booking_pk):
    """Displays a success page after a booking is confirmed."""
    booking = get_object_or_404(Booking, pk=booking_pk, user=request.user)
    if booking.user != request.user:
        messages.error(request, "You are not authorized to view this booking.")
        return redirect('my_bookings')

    return render(request, 'temple_app/booking_success.html', {'booking': booking})

Tabnine | Edit | Test | Explain | Document
@login_required
def my_bookings_view(request):
    """Displays a list of all bookings made by the logged-in user."""
    bookings = Booking.objects.filter(user=request.user).order_by('-booking_date')
    return render(request, 'temple_app/my_bookings.html', {'bookings': bookings})

Tabnine | Edit | Test | Explain | Document
@login_required
def add_temple_view(request):
    """Handles adding a new temple via a form (admin/staff only typically)."""
    if request.method == 'POST':
        form = TempleForm(request.POST, request.FILES)
        if form.is_valid():
```

```
    if booking.status == 'cancelled':
        messages.info(request, "This booking has already been cancelled.")
        return redirect('my_bookings')

    if request.method == 'POST':
        with transaction.atomic():
            booking_to_cancel = Booking.objects.select_for_update().get(pk=booking_pk)

            if booking_to_cancel.status == 'confirmed':
                booking_to_cancel.status = 'cancelled'
                booking_to_cancel.save()

                ticket_type = booking_to_cancel.ticket_type
                ticket_type.available_tickets += booking_to_cancel.quantity
                ticket_type.save()

                messages.success(request, f"Booking {booking_pk} has been cancelled successfully. Tickets refunded.")
            else:
                messages.error(request, f"Booking {booking_pk} cannot be cancelled as it's not in 'confirmed' status.")
        return redirect('my_bookings')

    return render(request, 'temple_app/cancel_booking_confirm.html', {'booking': booking})


# --- User Authentication Views ---
class RegisterView(CreateView):
    """View for user registration using a custom form."""
    form_class = CustomUserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'registration/register.html'

    Tabnine | Edit | Test | Explain | Document
    def form_valid(self, form):
```

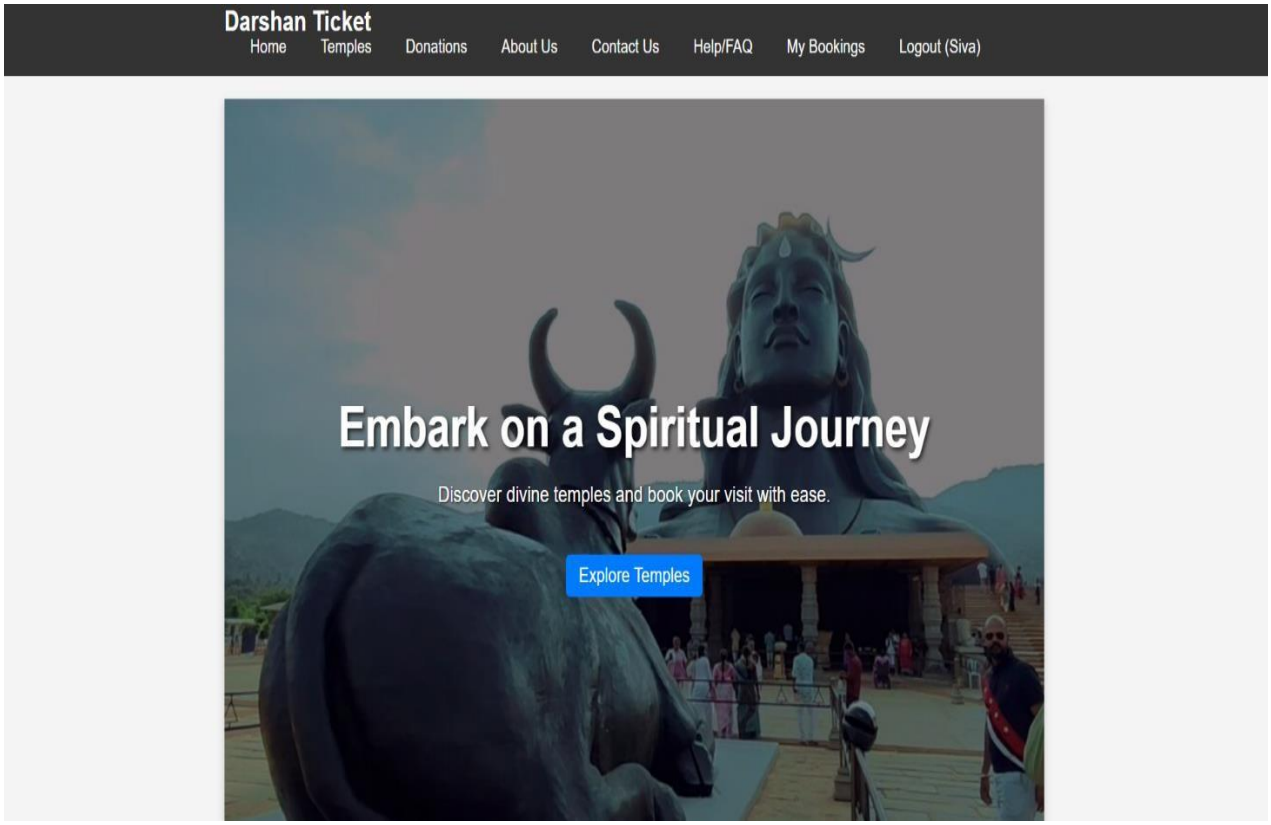
```
        response = super().form_valid(form)
        messages.success(self.request, 'Registration successful! Please log in.')
        return response


# --- Static Content & Contact Views ---

Tabnine | Edit | Test | Explain | Document
def about_us_view(request):
    """Renders the about us page."""
    return render(request, 'temple_app/about.html')

Tabnine | Edit | Test | Explain | Document
def contact_us_view(request):
    """Handles the contact us form submission and renders the page."""
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            name = form.cleaned_data['name']
            from_email = form.cleaned_data['email']
            subject = form.cleaned_data['subject']
            message = form.cleaned_data['message']

            full_message = f"Message from: {name} ({from_email})\n\nSubject: {subject}\n\n{message}"

            try:
                send_mail(
                    subject=f"Feedback: {subject}",
                    message=full_message,
                    from_email=settings.DEFAULT_FROM_EMAIL,
                    recipient_list=['your_email@example.com'], # <--- REMEMBER TO REPLACE THIS WITH YOUR ACTUAL EMAIL
                    fail_silently=False,
                )
                messages.success(request, 'Your message has been sent successfully!')
```

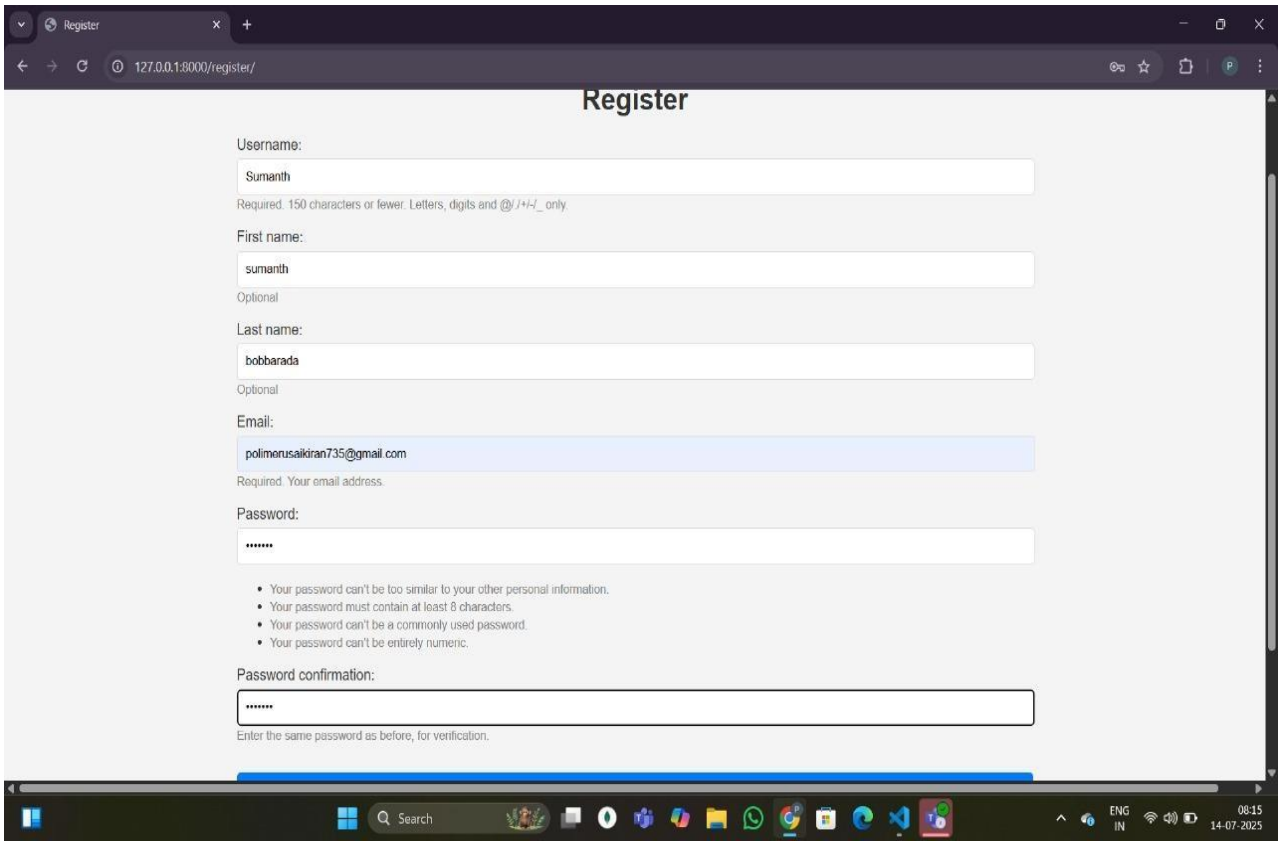
```
# --- Donation Views---

Tabnine | Edit | Test | Explain | Document
def donations_view(request):
    """
    Renders the donation page with the form and initial donation statistics.
    Uses the 'donations.html' template.
    """
    form = DonationForm() # No 'user' argument here, it's handled in create_donation_ajax

    total_donations_amount = Donation.objects.aggregate(Sum('amount'))['amount__sum'] or 0.00
    recent_donations = Donation.objects.all().order_by('-timestamp')[:5]

    context = {
        'form': form,
        'total_donations': total_donations_amount,
        'recent_donations': recent_donations,
    }
    # Pointing to the new donations.html
    return render(request, 'temple_app/donations.html', context)


Tabnine | Edit | Test | Explain | Document
@require_POST # Ensures only POST requests are processed
def create_donation_ajax(request):
    """
    API endpoint to handle AJAX submission of donation form data.
    Uses FormData, so data is in request.POST.
    """
    form = DonationForm(request.POST)

    if form.is_valid():
        donation = form.save(commit=False)

        # If user is authenticated, link the donation to them
        if request.user.is_authenticated:
```

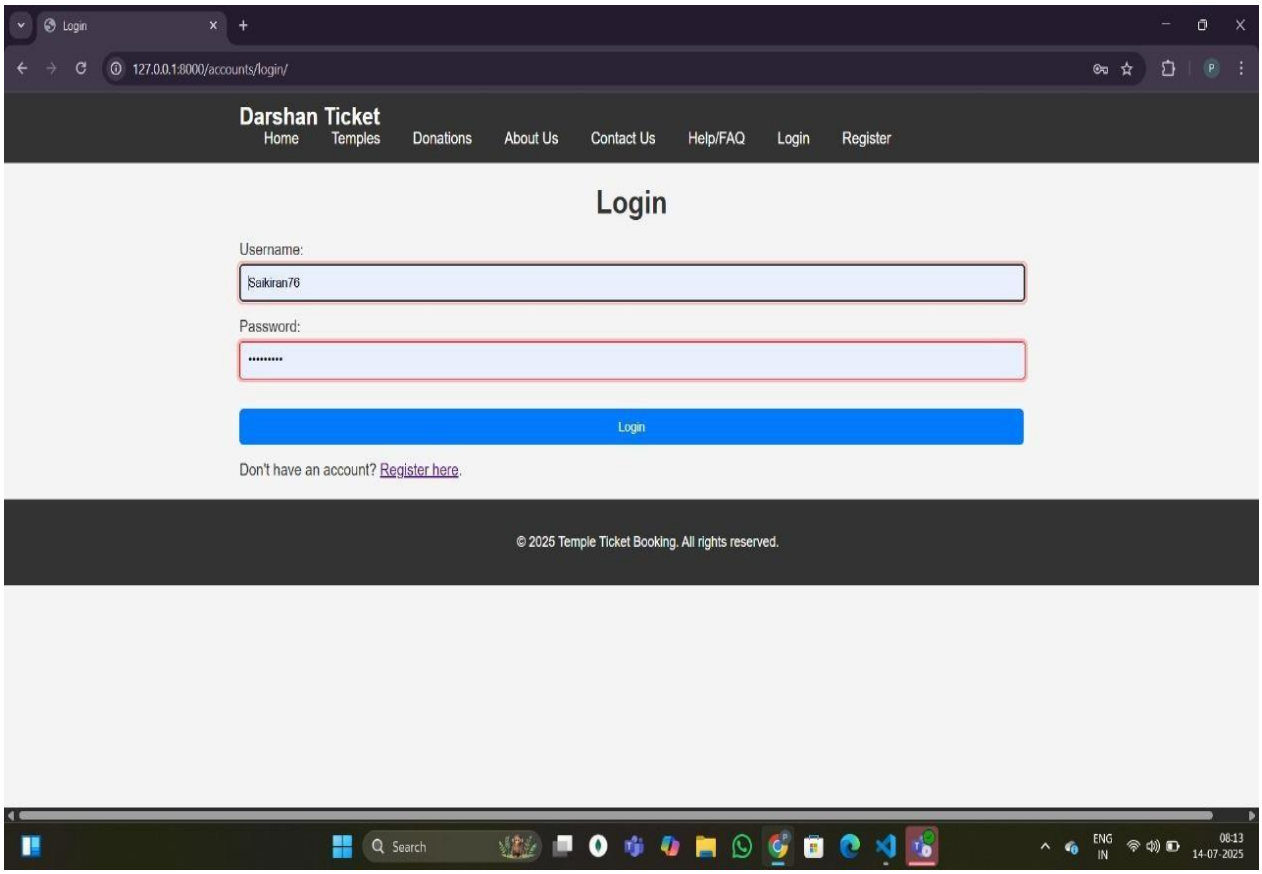
```
    """
    form = DonationForm(request.POST)

    if form.is_valid():
        donation = form.save(commit=False)

        # If user is authenticated, link the donation to them
        if request.user.is_authenticated:
            donation.user = request.user
            # If donor_name is not explicitly provided, default to user's name
            if not donation.donor_name:
                donation.donor_name = request.user.get_full_name() or request.user.username

        # If anonymous is checked, ensure donor_name is null regardless of user
        if donation.is_anonymous:
            donation.donor_name = None

        donation.save()

        return JsonResponse({'status': 'success', 'message': 'Thank you for your generous donation!'})
    else:
        # Form is not valid, return errors in a format the JS can parse
        return JsonResponse({
            'status': 'error',
            'message': 'Please correct the errors in the form.',
            'errors': form.errors.as_json() # Returns a JSON string of errors
        }, status=400)


Tabnine | Edit | Test | Explain | Document
def api_donations_data(request):
    """
    API endpoint to provide total and recent donation data for AJAX updates.
    """
    total_donations_amount = Donation.objects.aggregate(Sum('amount'))['amount__sum'] or 0.00
```

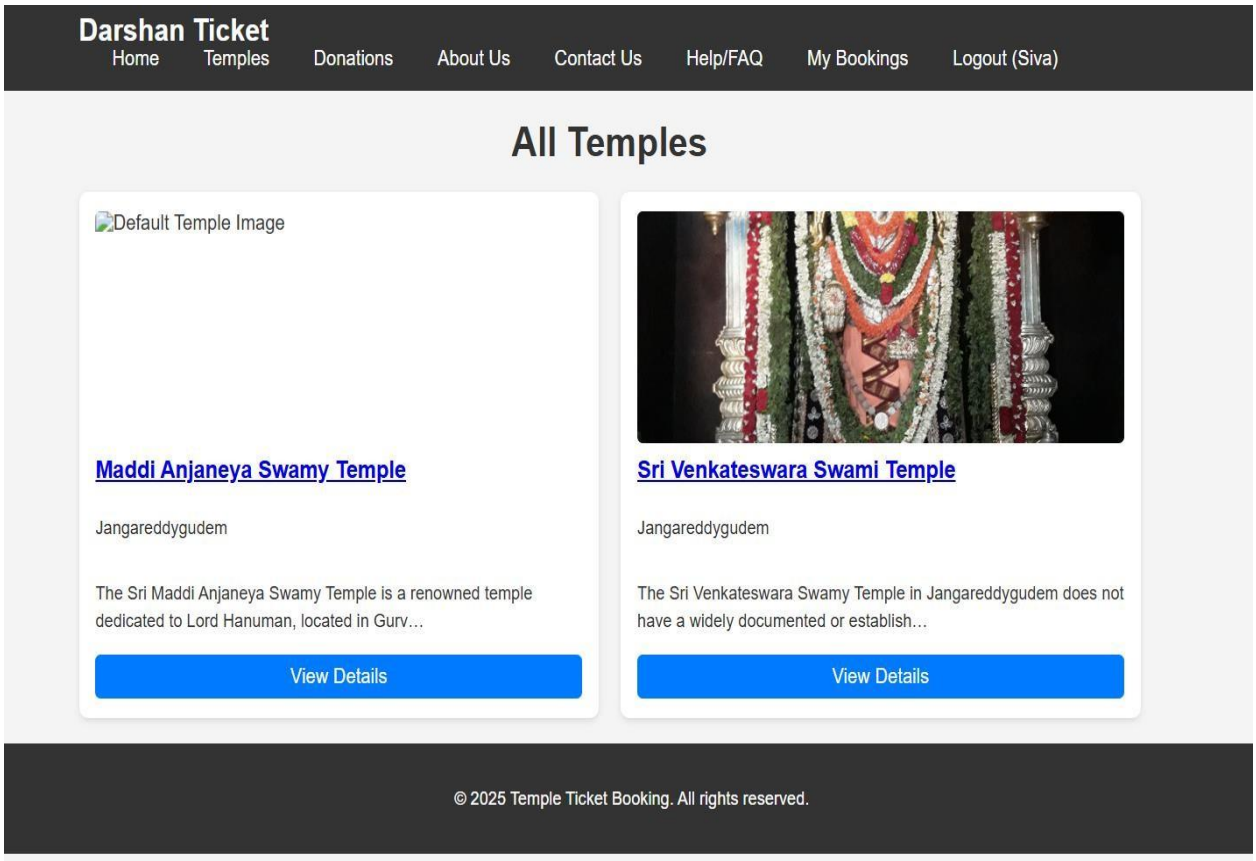
```
Tabnine | Edit | Test | Explain | Document
def api_donations_data(request):
    """
    API endpoint to provide total and recent donation data for AJAX updates.
    """
    total_donations_amount = Donation.objects.aggregate(Sum('amount'))['amount__sum'] or 0.00
    recent_donations_queryset = Donation.objects.all().order_by('-timestamp')[:5]

    recent_donations_list = []
    for donation in recent_donations_queryset:
        # Convert model instance to dictionary for easy JSON serialization
        data = model_to_dict(donation, fields=['amount', 'is_anonymous', 'message'])

        # Override donor_name logic for display based on template's needs
        if donation.is_anonymous:
            data['donor_name'] = 'Anonymous'
        elif donation.donor_name:
            data['donor_name'] = donation.donor_name
        elif donation.user and donation.user.username:
            data['donor_name'] = donation.user.get_full_name() or donation.user.username
        else:
            data['donor_name'] = 'N/A' # Fallback if no name or user

        # Format timestamp for display in JS
        data['timestamp'] = donation.timestamp.strftime("%b %d, %Y %H:%M")
        recent_donations_list.append(data)

    return JsonResponse({
        'total_donations': total_donations_amount,
        'recent_donations': recent_donations_list
    })
```

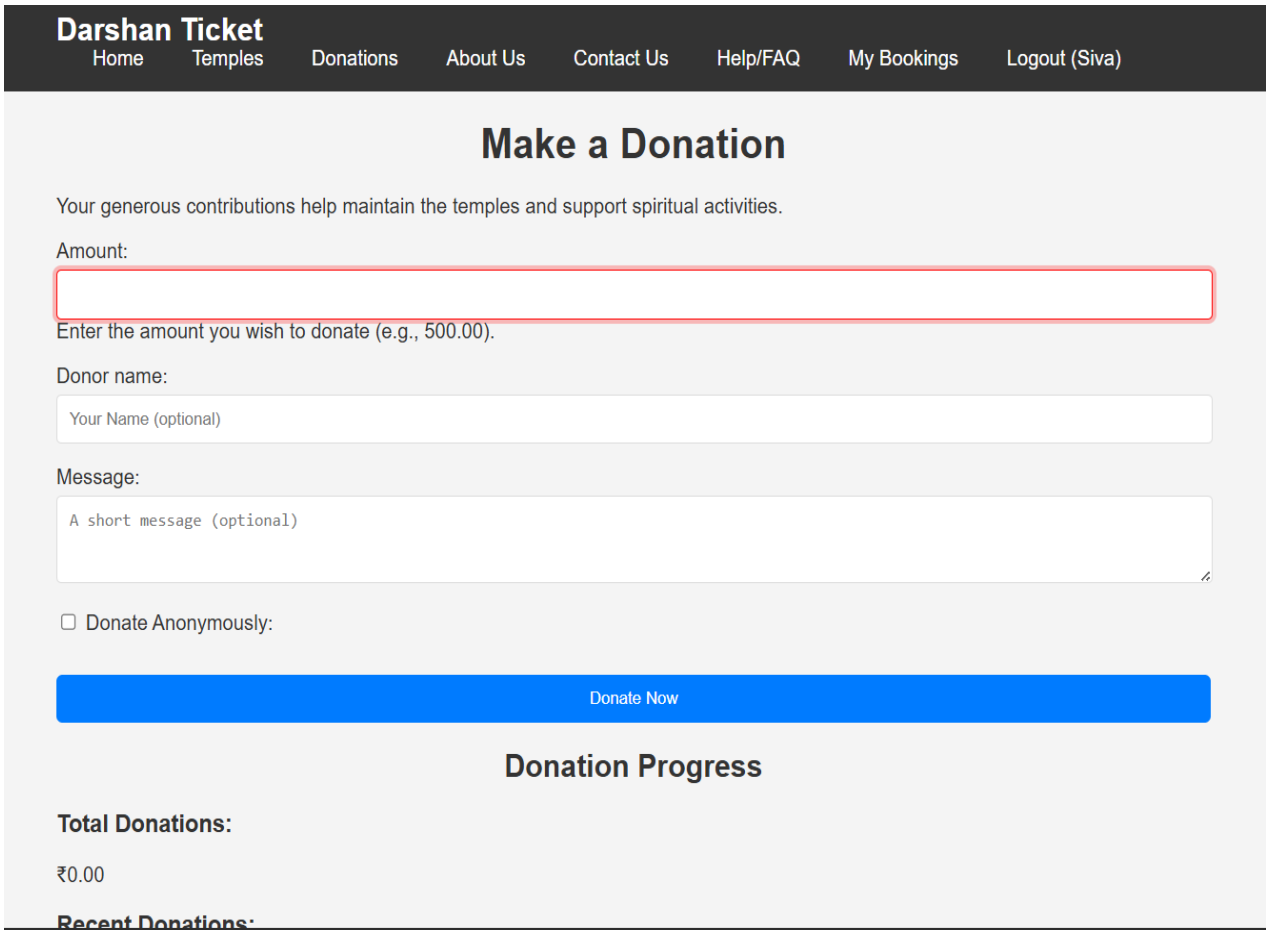
OUTPUT IMAGES

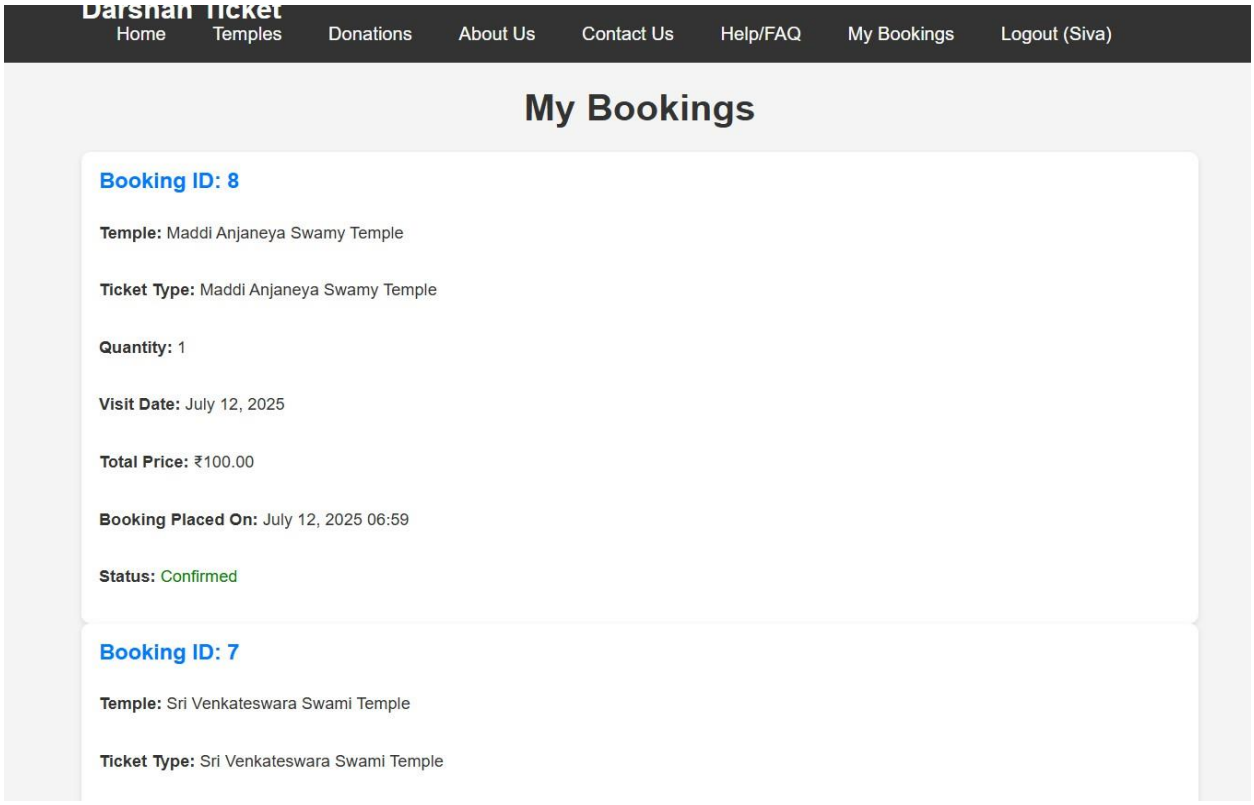





Login
127.0.0.1:8000/accounts/login/
Darshan Ticket
Home
Temples
Donations
About Us
Contact Us
Help/FAQ
Login
Register
Login
Username:
Saikiran76
Password:
Login
Don't have an account? Register here.
© 2025 Temple Ticket Booking. All rights reserved.
Search
ENG
IN
08:13
14-07-2025


Darshan Ticket
Home
Temples
Donations
About Us
Contact Us
Help/FAQ
My Bookings
Logout (Siva)
All Temples
Default Temple Image
Maddi Anjaneya Swamy Temple
Jangareddygudem
The Sri Maddi Anjaneya Swamy Temple is a renowned temple dedicated to Lord Hanuman, located in Gurv…
View Details
Sri Venkateswara Swami Temple
Jangareddygudem
The Sri Venkateswara Swamy Temple in Jangareddygudem does not have a widely documented or establish…
View Details
© 2025 Temple Ticket Booking. All rights reserved.


Darshan Ticket
Home
Temples
Donations
About Us
Contact Us
Help/FAQ
My Bookings
Logout (Siva)
Make a Donation
Your generous contributions help maintain the temples and support spiritual activities.
Amount:
Enter the amount you wish to donate (e.g., 500.00).
Donor name:
Your Name (optional)
Message:
A short message (optional)
Donate Anonymously:
Donate Now
Donation Progress
Total Donations:
₹0.00
Recent Donations:


Darshan Ticket
Home
Temples
Donations
About Us
Contact Us
Help/FAQ
My Bookings
Logout (Siva)
My Bookings
Booking ID: 8
Temple: Maddi Anjaneya Swamy Temple
Ticket Type: Maddi Anjaneya Swamy Temple
Quantity: 1
Visit Date: July 12, 2025
Total Price: ₹100.00
Booking Placed On: July 12, 2025 06:59
Status: Confirmed
Booking ID: 7
Temple: Sri Venkateswara Swami Temple
Ticket Type: Sri Venkateswara Swami Temple

Embrace the Divine Presence

Lord Shiva

Goddess Parvati

Lord Vishnu

Goddess lakshmi

Featured Temples

Default Temple Image

Maddi Anjaneya Swamy Temple

Sri Venkateswara Swami Temple

CONCLUSION

This project successfully developed a user-friendly Temple Ticket Booking System using Django. It streamlines temple visit planning by allowing users to register, log in, book tickets, view their bookings, and make donations online. The admin panel enables temple authorities to manage bookings, ticket types, and temple details efficiently.

By reducing long queues and manual work, the system offers a centralized platform that is secure, scalable, and easy to use. It shows how technology can enhance religious services and improve user experience. Future improvements could include online payments, notifications, and mobile app support—making temple visits even more accessible and organized.

REFERENCES

- **Django Official Documentation — Complete reference for Django features, including models, views, templates, URL routing, authentication, and session management.**
  https://docs.djangoproject.com/en/5.2/

- **Django Project Examples on GitHub — Open-source Django projects with booking functionality, login/register, and admin dashboards—great for learning best practices.**
  https://github.com/search?q=temple+booking+django

- **Bootstrap Framework for Frontend Design — Helps build responsive and mobile-friendly UI for forms, navbars, and booking pages in your Django templates.**
  https://getbootstrap.com/

- **Customizing Django Admin Panel — Guide to extend Django's built-in admin for managing temple data, ticket types, and booking records efficiently.**
  https://simpleisbetterthancomplex.com/tutorial/2016/07/18/how-to-extend-django-admin.html

- **Indian Religious Places Dataset — Basic info on temples in India, available from government sources; useful for testing or seeding your application.**
  https://data.gov.in/catalog/religious-places-temples

- **Deployment of Django Projects — Covers hosting your Django application using platforms like Heroku, PythonAnywhere, or Render with production-ready setup.**
  https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Deployment

- **Razorpay Django Integration — A detailed guide on adding a payment gateway for online donations or paid temple services.**
  https://razorpay.com/docs/payment-gateway/server-integration/django/

- **Django Forms & Validation — How to build and validate user-friendly forms for login, registration, and ticket booking with Django's built-in features.**
  https://docs.djangoproject.com/en/5.2/ref/forms/

- **Django Security Best Practices — Guidelines on securing your Django web app using session handling, CSRF protection, and safe user input validation.**
  https://docs.djangoproject.com/en/5.2/topics/security/