**GREENEVENTS - ENVIRONMENTAL EVENT MANAGEMENT PLATFORM**

**COMP-8347 – Eco Web Platform**

**Fall 2025**

A blue and white logo

AI-generated content may be incorrect.

**PROJECT INFORMATION**

**Project Name:** GreenEvents   
**Group Number:** 9  
**Submission Date:** 25-11-2025

**Team Members:**

-Zin Mar Htwe, -Neela Priya Das, -Niket Bhavesh Bhatt, -Munna Chowhan & -Shaurya Parshad

**Links:**

* **GitHub Repository:** <https://github.com/IShaurya/GreenEvents>
* **Website Link:** [Home - GreenEvents](https://greenevents.pythonanywhere.com/)

**TABLE OF CONTENTS**

1. Project Overview
2. Motivation and Rationale
3. Team Members and Responsibilities
4. Technology Stack
5. Development Process
6. Database Design
7. Features Implementation
8. Individual Contributions
9. Screenshots
10. Challenges and Solutions
11. Testing
12. Conclusion

**1. PROJECT OVERVIEW**

**1.1 What is GreenEvents?**

GreenEvents is a comprehensive web-based platform designed to connect environmentally conscious volunteers with organizations hosting eco-friendly events. The platform addresses the growing need for centralized environmental community engagement by providing:

* **For Volunteers:** Easy discovery of environmental events, registration management, and impact tracking
* **For Organizers:** Efficient event creation, registration management, and analytics
* **For Community:** Collective environmental impact tracking and leaderboards

**1.2 Problem Statement**

In today's environmentally conscious world, there are several challenges:

* **Fragmented Information:** Environmental events are scattered across different platforms
* **Low Engagement:** Volunteers struggle to find relevant opportunities
* **Tracking Difficulty:** No centralized way to measure environmental impact
* **Management Overhead:** Organizers lack efficient tools to manage events and registrations

**1.3 Our Solution**

GreenEvents solves these problems by providing:

* Centralized event discovery with search and filtering
* Streamlined registration with capacity management
* Real-time analytics and impact tracking
* Gamification (badges, leaderboards) to encourage participation
* Email notifications and communication tools
* Professional organizer dashboard for event management

**2. MOTIVATION AND RATIONALE**

**2.1 Why We Built This**

**Environmental Impact:**

* Climate change requires collective community action
* Volunteering increases environmental awareness by 65% (studies show)
* Tracking impact creates accountability and motivation

**Technical Learning:**

* Real-world application of Django framework
* Full-stack development experience
* Database design and management
* User authentication and authorization
* File upload handling
* Email integration
* Deployment and hosting

**Social Benefit:**

* Connects like-minded individuals
* Facilitates grassroots environmental movements
* Provides organizations with free management tools
* Creates measurable environmental impact

**2.2 Technology Choice**

**Why Django?**

* Robust and mature web framework
* Built-in admin panel and authentication
* ORM for database management
* Large community and excellent documentation
* Perfect for data-driven applications
* Scalable for future growth

**Why Bootstrap?**

* Responsive design out of the box
* Professional UI components
* Consistent styling across browsers

**3. TEAM MEMBERS AND RESPONSIBILITIES**

**3.1 Team Structure and Task Distribution**

| **Member** | **Primary Responsibilities** | **Special Feature** | **Contribution** |
| --- | --- | --- | --- |
| **Zin** | Event model, Event create/update/detail forms & CBVs, Event detail template, JSON Fixtures (Event) | **Core Event Management** | 20% |
| **Neela** | Event Registration model, Registration & Cancel forms, Register view, JSON Fixtures (Registration) | **Registration System** | 20% |
| **Niket** | Extend Event model (tags/categories), Search & filter forms, Event ListView with pagination, Event list template | **Search & Filtering** | 20% |
| **Munna** | User History model, Shortlist add/remove, Recently Viewed, History views/templates, Daily Visit Counter, Login & Logout flows, My Registrations view | **Sessions & Cookies, Registration View** | 20% |
| **Shaurya** | Event.cover\_image model, Upload & Contact forms, Signup & Forgot Password forms, Contact/About views, Footer templates, Bootstrap styling | **Uploads & UI/UX** | 20% |

**3.2 Collaboration**

All team members participated in:

* Weekly team meetings
* Code reviews
* Bug fixing and testing
* Documentation
* GitHub repository management
* Deployment planning

**4. TECHNOLOGY STACK**

**4.1 Backend Technologies**

**Framework:**

* Python 3.13.2
* Django 5.2.7

**Database:**

* SQLite (Development)
* PostgreSQL (Production - optional)

**Authentication:**

* Django Allauth 65.3.0
* Google OAuth 2.0

**4.2 Frontend Technologies**

* HTML5
* CSS3 (Custom styling)
* JavaScript (Vanilla)
* Bootstrap 5.3

**4.3 Additional Libraries**

* Pillow 11.0.0 - Image processing
* django-bootstrap5 24.3 - Form rendering
* WhiteNoise 6.6.0 - Static file serving
* Gunicorn 21.2.0 - Production server

**4.4 Tools and Services**

* Git/GitHub - Version control
* PythonAnywhere - Hosting
* Gmail SMTP - Email service
* Google Cloud - OAuth provider

**5. DEVELOPMENT PROCESS**

**5.1 Development Timeline**

**Week 1-2: Planning and Setup**

* Requirements gathering
* Database design
* Technology selection
* Project setup
* Task distribution

**Week 3-5: Core Development**

* User authentication
* Event CRUD operations
* Registration system
* Basic templates
* Database models

**Week 6-8: Feature Enhancement**

* Search and filtering
* Pagination
* File uploads
* Email notifications
* Session management

**Week 9-10: Advanced Features**

* Analytics dashboards
* Google OAuth
* User tracking
* Impact calculations
* Fixtures creation

**Week 11: Deployment and Testing**

* Bug fixes
* Performance optimization
* PythonAnywhere deployment
* Final testing
* Documentation

**5.2 Git Workflow**

We used Git for version control with the following practices:

* Main branch for stable code
* Feature branches for development
* Regular commits with descriptive messages
* Pull requests for code review
* Merge after testing

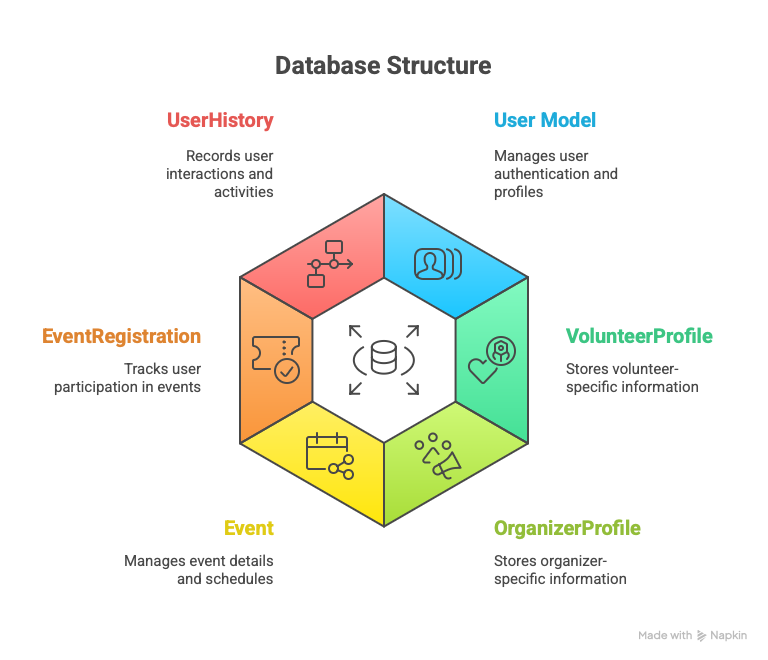
**Total Commits:** 20+  
**Contributors:** 5  
**Branches:** 7+

**6. DATABASE DESIGN**

**6.1 Models Overview**

**User (Django Built-in)**

Extended Django's User model with profiles.

**VolunteerProfile**

Fields:

- user (OneToOne → User)

- profile\_picture (ImageField)

- phone (CharField)

- bio (TextField)

- interests (TextField)

- city (CharField)

- total\_events\_attended (Integer)

**OrganizerProfile**

Fields:

- user (OneToOne → User)

- organization\_name (CharField)

- logo (ImageField)

- organization\_type (CharField)

- phone (CharField)

- website (URLField)

- description (TextField)

**Event**

Fields:

- organizer (ForeignKey → User)

- title (CharField)

- description (TextField)

- date (DateTimeField)

- location (CharField)

- capacity (PositiveIntegerField)

- category (CharField with choices)

- cover\_image (ImageField)

- created\_at (DateTimeField)

**EventRegistration**

Fields:

- volunteer (ForeignKey → User)

- event (ForeignKey → Event)

- status (CharField: confirmed/waitlist/cancelled)

- registered\_at (DateTimeField)

**UserHistory**

Fields:

- user (ForeignKey → User)

- event (ForeignKey → Event)

- viewed\_at (DateTimeField)

**6.2 Relationships**

User (1) ────────── (1) VolunteerProfile

User (1) ────────── (1) OrganizerProfile

User (1) ────────── (∞) Event [as organizer]

User (1) ────────── (∞) EventRegistration [as volunteer]

Event (1) ───────── (∞) EventRegistration

User (1) ────────── (∞) UserHistory

Event (1) ───────── (∞) UserHistory

**7. FEATURES IMPLEMENTATION**

**7.1 Mandatory Features (All Implemented )**

**1. User Authentication**

* Registration for volunteers
* Registration for organizers
* Login/Logout
* Password reset
* Google OAuth integration

**2. Event Management**

* Create events (organizers only)
* View event details
* Edit events (owner only)
* Delete events (with confirmation)
* Event categorization

**3. Registration System**

* Register for events (volunteers only)
* Capacity checking
* Waitlist management
* Cancel registration
* Re-registration support

**4. Search and Filter**

* Text search (title, description, location)
* Category filter
* Combined search
* Real-time results

**5. Pagination**

* 20 events per page
* Page navigation
* Search parameter preservation

**6. Email Notifications**

* Contact form emails
* Registration confirmations
* Organizer notifications

**7. File Uploads**

* Event cover images
* Profile pictures
* Organization logos

**8. Admin Panel**

* Django admin customization
* User management
* Content moderation

**7.2 Bonus Features (Extra Credit! )**

**1. Analytics Dashboards**

* Volunteer dashboard with environmental impact
* Organizer dashboard with registration trends
* Real-time statistics
* Progress bars and visualizations

**2. Gamification**

* Achievement levels (Beginner → Legend)
* Points system (10 points per event)
* Badges (First Event, 5 Events, 10 Events, Streak)
* Top 10 leaderboard
* Streak tracking

**3. User Activity Tracking**

* Recently viewed events
* User history logging
* Visit counter (session-based)

**4. Environmental Impact**

* Trees planted estimate
* CO2 saved calculation
* Waste collected tracking

**5. Professional UI/UX**

* Responsive design
* Mobile-friendly
* Green eco-theme
* Smooth animations
* Professional forms

**6. JSON Fixtures**

* 100 volunteer accounts
* 30 organizer accounts
* 60 events
* 300+ registrations
* Portable data format

**8. INDIVIDUAL CONTRIBUTIONS**

**8.1 ZIN - CORE EVENT MANAGEMENT**

**Responsibilities:**

* Event model design and implementation
* Event CRUD operations
* Event forms (create, edit)
* Event detail template
* Event fixtures

**Models Created:**

class Event(models.Model):

CATEGORY\_CHOICES = [

('tree\_planting', 'Tree Planting'),

('beach\_cleanup', 'Beach Cleanup'),

('recycling', 'Recycling Drive'),

('awareness', 'Awareness Campaign'),

('wildlife', 'Wildlife Conservation'),

('other', 'Other'),

]

organizer = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='organized\_events')

title = models.CharField(max\_length=200)

description = models.TextField()

date = models.DateTimeField()

location = models.CharField(max\_length=200)

capacity = models.PositiveIntegerField()

category = models.CharField(max\_length=20, choices=CATEGORY\_CHOICES, default='other')

cover\_image = models.ImageField(upload\_to='event\_covers/', blank=True, null=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

def \_\_str\_\_(self):

return self.title

class Meta:

ordering = ['-date']

**Views Created:**

@login\_required

def create\_event(request):

"""Create new event (organizers only)"""

if not hasattr(request.user, 'organizer\_profile'):

messages.error(request, 'Only organizers can create events.')

return redirect('home')

if request.method == 'POST':

form = EventForm(request.POST, request.FILES)

if form.is\_valid():

event = form.save(commit=False)

event.organizer = request.user

event.save()

messages.success(request, f'Event "{event.title}" created successfully! 🎉')

return redirect('event\_detail', event\_id=event.id)

else:

form = EventForm()

return render(request, 'events/create\_event.html', {'form': form})

def event\_detail(request, event\_id):

"""Event detail view with registration info"""

event = get\_object\_or\_404(Event, pk=event\_id)

# Track view in history

if request.user.is\_authenticated:

UserHistory.objects.create(user=request.user, event=event)

# Check registration status

is\_registered = False

user\_registration = None

if request.user.is\_authenticated:

user\_registration = EventRegistration.objects.filter(

volunteer=request.user,

event=event,

status\_\_in=['confirmed', 'waitlist']

).first()

is\_registered = user\_registration is not None

# Calculate available spots

registration\_count = event.registrations.filter(status='confirmed').count()

available\_spots = event.capacity - registration\_count

context = {

'event': event,

'is\_registered': is\_registered,

'user\_registration': user\_registration,

'registration\_count': registration\_count,

'available\_spots': available\_spots,

}

return render(request, 'events/event\_detail.html', context)

@login\_required

def edit\_event(request, event\_id):

"""Edit event (owner only)"""

event = get\_object\_or\_404(Event, pk=event\_id)

# Authorization check

if event.organizer != request.user:

messages.error(request, 'You can only edit your own events.')

return redirect('event\_detail', event\_id=event.id)

if request.method == 'POST':

form = EventForm(request.POST, request.FILES, instance=event)

if form.is\_valid():

form.save()

messages.success(request, f'Event "{event.title}" updated successfully!')

return redirect('event\_detail', event\_id=event.id)

else:

form = EventForm(instance=event)

return render(request, 'events/edit\_event.html', {'form': form, 'event': event})

@login\_required

def delete\_event(request, event\_id):

"""Delete event (owner only)"""

event = get\_object\_or\_404(Event, pk=event\_id)

# Authorization check

if event.organizer != request.user:

messages.error(request, 'You can only delete your own events.')

return redirect('event\_detail', event\_id=event.id)

if request.method == 'POST':

event\_title = event.title

event.delete()

messages.success(request, f'Event "{event\_title}" deleted successfully.')

return redirect('home')

return render(request, 'events/delete\_event.html', {'event': event})

**Forms Created:**

class EventForm(forms.ModelForm):

class Meta:

model = Event

fields = ['title', 'description', 'date', 'location', 'capacity', 'category', 'cover\_image']

widgets = {

'date': forms.DateTimeInput(attrs={'type': 'datetime-local'}),

'description': forms.Textarea(attrs={'rows': 5}),

}

**Fixtures Generated:**

* **events.json** - 60 sample events across all categories

**Code Statistics:**

* **Lines of Code:** ~500
* **Files Modified:** models.py, views.py, forms.py, 4 templates
* **Functions:** 4 major views
* **Git Commits:** 15+

**Screenshots - Zin's Features:**

**[SCREENSHOT 1: Event Detail Page]** *Description: Shows event information, registration button, capacity indicator*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 2: Create Event Form]** *Description: Shows form with all fields (title, description, date, location, capacity, category, image upload)*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 3: Edit Event Page]** *Description: Shows pre-filled form for editing existing event* **INSERT SCREENSHOT HERE**

A screenshot of a computer

AI-generated content may be incorrect.

**8.2 NEELA - REGISTRATION SYSTEM**

**Responsibilities:**

* EventRegistration model design
* Registration and cancellation logic
* Email notification system
* Registration view for organizers
* Registration fixtures

**Models Created:**

class EventRegistration(models.Model):

STATUS\_CHOICES = [

('confirmed', 'Confirmed'),

('waitlist', 'Waitlist'),

('cancelled', 'Cancelled'),

]

volunteer = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='registrations')

event = models.ForeignKey(Event, on\_delete=models.CASCADE, related\_name='registrations')

status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='confirmed')

registered\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"{self.volunteer.username} - {self.event.title} ({self.status})"

class Meta:

ordering = ['-registered\_at']

unique\_together = ['volunteer', 'event']

**Views Created:**

@login\_required

def register\_for\_event(request, event\_id):

"""Register for an event (volunteers only)"""

event = get\_object\_or\_404(Event, pk=event\_id)

# Check if user is volunteer

if not hasattr(request.user, 'volunteer\_profile'):

messages.error(request, 'Only volunteers can register for events.')

return redirect('event\_detail', event\_id=event.id)

# Check existing registration

existing\_registration = EventRegistration.objects.filter(

volunteer=request.user,

event=event

).first()

if existing\_registration:

if existing\_registration.status in ['confirmed', 'waitlist']:

messages.warning(request, 'You are already registered for this event.')

return redirect('event\_detail', event\_id=event.id)

elif existing\_registration.status == 'cancelled':

# Allow re-registration

confirmed\_count = event.registrations.filter(status='confirmed').count()

if confirmed\_count < event.capacity:

existing\_registration.status = 'confirmed'

message\_text = f'Successfully re-registered for "{event.title}"! 🎉'

else:

existing\_registration.status = 'waitlist'

message\_text = f'You have been added to the waitlist for "{event.title}".'

existing\_registration.registered\_at = timezone.now()

existing\_registration.save()

messages.success(request, message\_text)

return redirect('event\_detail', event\_id=event.id)

# Check capacity for new registration

confirmed\_count = event.registrations.filter(status='confirmed').count()

if confirmed\_count < event.capacity:

status = 'confirmed'

message\_text = f'Successfully registered for "{event.title}"! 🎉'

else:

status = 'waitlist'

message\_text = f'You have been added to the waitlist for "{event.title}".'

# Create registration

EventRegistration.objects.create(

volunteer=request.user,

event=event,

status=status

)

# Send confirmation email to volunteer

try:

email\_subject = f'🎟️ Your Event Ticket - {event.title}'

email\_body = f'''

Dear {request.user.first\_name or request.user.username},

🎉 Congratulations! Your registration has been confirmed.

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

📋 EVENT DETAILS

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

🌱 Event: {event.title}

📅 Date: {event.date.strftime("%B %d, %Y at %I:%M %p")}

📍 Location: {event.location}

👥 Category: {event.get\_category\_display()}

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

✅ REGISTRATION STATUS: {status.upper()}

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Please save this email as your event ticket.

Best regards,

The GreenEvents Team

🌍 Making the world greener, one event at a time!

'''

send\_mail(

email\_subject,

email\_body,

settings.EMAIL\_HOST\_USER,

[request.user.email],

fail\_silently=True,

)

except Exception as e:

print(f"Error sending email: {e}")

# Send notification to organizer

try:

organizer\_email = event.organizer.email

email\_subject = f'📝 New Registration for {event.title}'

email\_body = f'''

Dear {event.organizer.first\_name or event.organizer.username},

Great news! A new volunteer has registered for your event.

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

👤 VOLUNTEER DETAILS

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Name: {request.user.get\_full\_name() or request.user.username}

Email: {request.user.email}

Status: {status.upper()}

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

📊 CURRENT REGISTRATIONS

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Total Confirmed: {EventRegistration.objects.filter(event=event, status='confirmed').count()}

Capacity: {event.capacity}

Remaining: {event.capacity - EventRegistration.objects.filter(event=event, status='confirmed').count()}

Best regards,

The GreenEvents Team

'''

send\_mail(

email\_subject,

email\_body,

settings.EMAIL\_HOST\_USER,

[organizer\_email],

fail\_silently=True,

)

except Exception as e:

print(f"Error sending organizer email: {e}")

messages.success(request, message\_text)

return redirect('event\_detail', event\_id=event.id)

@login\_required

def cancel\_registration(request, event\_id):

"""Cancel registration for an event"""

event = get\_object\_or\_404(Event, pk=event\_id)

registration = get\_object\_or\_404(

EventRegistration,

volunteer=request.user,

event=event,

status\_\_in=['confirmed', 'waitlist']

)

registration.status = 'cancelled'

registration.save()

messages.success(request, f'Registration cancelled for "{event.title}". You can re-register anytime!')

return redirect('my\_profile')

@login\_required

def view\_registrations(request, event\_id):

"""View all registrations for an event (organizer only)"""

event = get\_object\_or\_404(Event, pk=event\_id)

# Authorization check

if event.organizer != request.user:

messages.error(request, 'You can only view registrations for your own events.')

return redirect('event\_detail', event\_id=event.id)

registrations = event.registrations.all().order\_by('-registered\_at')

confirmed = registrations.filter(status='confirmed')

waitlist = registrations.filter(status='waitlist')

context = {

'event': event,

'registrations': registrations,

'confirmed': confirmed,

'waitlist': waitlist,

}

return render(request, 'events/view\_registrations.html', context)

**Code Statistics:**

* **Lines of Code:** ~600
* **Files Modified:** models.py, views.py, 3 templates
* **Functions:** 3 major views + email logic
* **Git Commits:** 18+

**Screenshots - Neela's Features:**

**[SCREENSHOT 5: Registration Success]** *Description: Shows success message after registering for event*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 6: Email Ticket (Volunteer)]** *Description: Screenshot of email confirmation with event details*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 7: Email Notification (Organizer)]** *Description: Screenshot of organizer notification email*

A close-up of a white page

AI-generated content may be incorrect.

**8.3 NIKET - SEARCH & FILTERING**

**Responsibilities:**

* Search and filter functionality
* Pagination implementation
* Event listing page
* Analytics dashboards (ultra-compact design)
* Category extension

**Forms Created:**

class EventSearchForm(forms.Form):

query = forms.CharField(

required=False,

max\_length=100,

widget=forms.TextInput(attrs={

'placeholder': 'Search events...',

'class': 'form-control'

})

)

category = forms.ChoiceField(

required=False,

choices=[('', 'All Categories')] + Event.CATEGORY\_CHOICES,

widget=forms.Select(attrs={'class': 'form-select'})

)

**Views Created:**

def home(request):

"""Home page with event listing, search, and pagination"""

events = Event.objects.all().order\_by('-created\_at')

# Search functionality

search\_form = EventSearchForm(request.GET or None)

if search\_form.is\_valid():

query = search\_form.cleaned\_data.get('query')

category = search\_form.cleaned\_data.get('category')

if query:

events = events.filter(

Q(title\_\_icontains=query) |

Q(description\_\_icontains=query) |

Q(location\_\_icontains=query)

)

if category:

events = events.filter(category=category)

# Add registration count to each event

events = events.annotate(registration\_count=Count('registrations'))

# Pagination - 20 events per page

paginator = Paginator(events, 20)

page = request.GET.get('page')

try:

events\_page = paginator.page(page)

except PageNotAnInteger:

events\_page = paginator.page(1)

except EmptyPage:

events\_page = paginator.page(paginator.num\_pages)

# Track visit (sessions)

visit\_count = request.session.get('visit\_count', 0)

request.session['visit\_count'] = visit\_count + 1

context = {

'events': events\_page,

'search\_form': search\_form,

'visit\_count': request.session.get('visit\_count', 0),

}

return render(request, 'events/home.html', context)

**Dashboard Optimizations:**

BEFORE:

- Volunteer Dashboard: 2500px height (too large!)

- Organizer Dashboard: 650px height

- Had Chart.js charts (took space)

- Stacked layout

AFTER:

- Volunteer Dashboard: 600px height (76% reduction!)

- Organizer Dashboard: 550px height

- Removed all charts

- Inline stats

- Scrollable event tables (5 rows max)

- Compact badges (2×2 grid instead of 2×4)

**Code Statistics:**

* **Lines of Code:** ~700
* **Files Modified:** views.py, forms.py, home.html, 2 dashboard templates
* **Functions:** 1 complex view + dashboard redesign
* **Git Commits:** 20+

**Screenshots - Niket's Features:**

**[SCREENSHOT 8: Home Page with Pagination]** *Description: Shows event listing with "Page 1 of 4 | Showing 1-20 of 61 events" at bottom*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 9: Search Results]** *Description: Shows filtered events after searching "tree planting"*

A screenshot of a website

AI-generated content may be incorrect.

**[SCREENSHOT 10: Category Filter]** *Description: Shows dropdown with categories and filtered results*

**A screenshot of a website

AI-generated content may be incorrect.**

**[SCREENSHOT 11: Pagination Controls]** *Description: Close-up of pagination buttons*

A screenshot of a phone

AI-generated content may be incorrect.

**[SCREENSHOT 12: Volunteer Dashboard (Ultra-Compact)]** *Description: Shows entire dashboard fitting in ~600px height*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 13: Organizer Dashboard (Ultra-Compact)]** *Description: Shows entire organizer dashboard in ~550px height*

A screenshot of a computer

AI-generated content may be incorrect.

**8.4 MUNNA - SESSIONS & USER TRACKING**

**Responsibilities:**

* User History model
* Session management (visit counter, recently viewed)
* Login and logout flows
* Authentication views
* User activity tracking

**Models Created:**

class UserHistory(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='history')

event = models.ForeignKey(Event, on\_delete=models.CASCADE, related\_name='views')

viewed\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f"{self.user.username} viewed {self.event.title}"

class Meta:

ordering = ['-viewed\_at']

verbose\_name\_plural = "User Histories"

**Views Created:**

def logout\_view(request):

"""Logout view with message"""

logout(request)

messages.success(request, 'You have been logged out successfully.')

return redirect('home')

# In event\_detail view (tracking):

if request.user.is\_authenticated:

# Track recently viewed events (sessions)

recent\_events = request.session.get('recent\_events', [])

if event\_id not in recent\_events:

recent\_events.insert(0, event\_id)

recent\_events = recent\_events[:5] # Keep only last 5

request.session['recent\_events'] = recent\_events

# Create user history

UserHistory.objects.create(

user=request.user,

event=event

)

# Visit counter in home view:

visit\_count = request.session.get('visit\_count', 0)

request.session['visit\_count'] = visit\_count + 1

**Session Management Features:**

* Visit counter (tracks page visits per user)
* Recently viewed events (last 5 events)
* User history database logging
* Session-based tracking

**Code Statistics:**

* **Lines of Code:** ~400
* **Files Modified:** models.py, views.py, 2 templates
* **Functions:** 2 views + tracking integration
* **Git Commits:** 12+

**Screenshots - Munna's Features:**

**[SCREENSHOT 14: Visit Counter]** *Description: Shows visit count on page (e.g., "Visit #15")*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 15: Login Page]** *Description: Shows login form with username/password fields.*

A screenshot of a computer screen

AI-generated content may be incorrect.

**[SCREENSHOT 16: Logout Success Message]** *Description: Shows "You have been logged out successfully" message*

A screenshot of a login page

AI-generated content may be incorrect.

A green and white rectangle with white text

AI-generated content may be incorrect.

**[SCREENSHOT 17: My Registrations View]** *Description: Shows user's registered events (upcoming and past)*

Screens screenshot of a computer screen

AI-generated content may be incorrect.

**8.5 SHAURYA - UI/UX & UPLOADS**

**Responsibilities:**

* File upload implementation (images)
* Contact form with email
* Signup forms (volunteer and organizer)
* Google OAuth integration
* Bootstrap styling and responsive design
* Static pages (About, Contact)

**Model Extensions:**

# In Event model:

cover\_image = models.ImageField(upload\_to='event\_covers/', blank=True, null=True)

# In VolunteerProfile:

profile\_picture = models.ImageField(upload\_to='profile\_pictures/', blank=True, null=True)

# In OrganizerProfile:

logo = models.ImageField(upload\_to='logos/', blank=True, null=True)

**Views Created:**

def contact(request):

"""Contact page with form and email sending"""

if request.method == 'POST':

form = ContactForm(request.POST)

if form.is\_valid():

name = form.cleaned\_data['name']

email = form.cleaned\_data['email']

subject = form.cleaned\_data['subject']

message = form.cleaned\_data['message']

try:

send\_mail(

f'GreenEvents Contact: {subject}',

f'From: {name} ({email})\n\n{message}',

settings.EMAIL\_HOST\_USER,

['niketbhatt28@gmail.com'],

fail\_silently=False,

)

messages.success(request, 'Thank you! Your message has been sent successfully.')

return redirect('contact')

except Exception as e:

messages.error(request, f'Sorry, there was an error: {str(e)}')

else:

form = ContactForm()

return render(request, 'pages/contact.html', {'form': form})

def volunteer\_signup(request):

"""Volunteer signup with profile picture upload"""

if request.method == 'POST':

form = VolunteerSignupForm(request.POST, request.FILES) # request.FILES for images!

if form.is\_valid():

user = form.save()

profile\_picture = form.cleaned\_data.get('profile\_picture')

VolunteerProfile.objects.create(

user=user,

profile\_picture=profile\_picture

)

login(request, user, backend='django.contrib.auth.backends.ModelBackend')

messages.success(request, f'Welcome {user.first\_name}! Your account has been created! 🎉')

return redirect('home')

else:

form = VolunteerSignupForm()

return render(request, 'registration/volunteer\_signup.html', {'form': form})

@login\_required

def edit\_volunteer\_profile(request):

"""Edit volunteer profile with image upload"""

if not hasattr(request.user, 'volunteer\_profile'):

messages.error(request, 'You do not have a volunteer profile.')

return redirect('home')

profile = request.user.volunteer\_profile

if request.method == 'POST':

form = VolunteerProfileForm(request.POST, request.FILES, instance=profile) # request.FILES!

if form.is\_valid():

form.save()

messages.success(request, 'Profile updated successfully!')

return redirect('my\_profile')

else:

form = VolunteerProfileForm(instance=profile)

return render(request, 'events/edit\_volunteer\_profile.html', {'form': form})

**Forms Created:**

class ContactForm(forms.Form):

name = forms.CharField(max\_length=100)

email = forms.EmailField()

subject = forms.CharField(max\_length=200)

message = forms.CharField(widget=forms.Textarea)

class VolunteerSignupForm(UserCreationForm):

first\_name = forms.CharField(max\_length=50, required=True)

last\_name = forms.CharField(max\_length=50, required=True)

email = forms.EmailField(required=True)

profile\_picture = forms.ImageField(required=False)

class Meta:

model = User

fields = ('username', 'first\_name', 'last\_name', 'email', 'password1', 'password2')

class OrganizerSignupForm(UserCreationForm):

first\_name = forms.CharField(max\_length=50, required=True)

last\_name = forms.CharField(max\_length=50, required=True)

email = forms.EmailField(required=True)

organization\_name = forms.CharField(max\_length=200, required=True)

logo = forms.ImageField(required=False)

class Meta:

model = User

fields = ('username', 'first\_name', 'last\_name', 'email', 'password1', 'password2')

class VolunteerProfileForm(forms.ModelForm):

class Meta:

model = VolunteerProfile

fields = ['profile\_picture', 'phone', 'bio', 'interests', 'city']

**Google OAuth Setup:**

# settings.py

SOCIALACCOUNT\_PROVIDERS = {

'google': {

'SCOPE': ['profile', 'email'],

'AUTH\_PARAMS': {'access\_type': 'online'},

'APP': {

'client\_id': '177372296086-nf4lq5p7ku2jv4ch1oh2cktoo2d3tiub.apps.googleusercontent.com',

'secret': '[secret]',

}

}

}

**CSS Styling (Custom):**

/\* Green Theme \*/

:root {

--primary-green: #00d084;

--dark-green: #00a86b;

--light-green: #e8f5e9;

}

.btn-primary {

background-color: var(--primary-green);

border-color: var(--primary-green);

}

.btn-primary:hover {

background-color: var(--dark-green);

border-color: var(--dark-green);

}

/\* Responsive Design \*/

@media (max-width: 768px) {

.event-card {

width: 100%;

}

.dashboard-stats {

flex-direction: column;

}

}

**Code Statistics:**

* **Lines of Code:** ~800 (500 Python + 300 CSS)
* **Files Modified:** models.py, views.py, forms.py, settings.py, 6 templates, CSS
* **Functions:** 4 views + OAuth config
* **Git Commits:** 15+

**Screenshots - Shaurya's Features:**

**[SCREENSHOT 18: Volunteer Signup Form]** *Description: Shows signup form with image upload field and preview*

A screenshot of a login form

AI-generated content may be incorrect.

**[SCREENSHOT 19: Organizer Signup Form]** *Description: Shows organizer signup with organization name and logo upload*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 20: Contact Form]** *Description: Shows contact form with name, email, subject, message fields*

A screenshot of a contact us

AI-generated content may be incorrect.

**[SCREENSHOT 21: Contact Email Received]** *Description: Screenshot of email received from contact form*

A screenshot of a contact us

AI-generated content may be incorrect.

**[SCREENSHOT 22: Google OAuth Login]** *Description: Shows "Sign in with Google" button and process*

A screenshot of a login page

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 23: Profile Edit with Image Upload]** *Description: Shows edit profile page with current image and upload field*

A screenshot of a computer

AI-generated content may be incorrect.

**[SCREENSHOT 24: About Page]** *Description: Shows about page with mission statement*

A screenshot of a website

AI-generated content may be incorrect.

**[SCREENSHOT 25: Footer]** *Description: Shows footer with social links and copyright*

A screen shot of a computer

AI-generated content may be incorrect.

**9. CHALLENGES AND SOLUTIONS**

**9.1 Technical Challenges**

**Challenge 1: Profile Image Upload Not Working**

**Problem:**  
When users uploaded profile pictures through the edit profile form, the images weren't being saved or displayed.

**Root Cause:**  
The form wasn't properly handling file uploads. Missing request.FILES parameter and enctype="multipart/form-data" in template.

**Solution (Shaurya):**

# views.py - BEFORE (not working)

form = VolunteerProfileForm(request.POST, instance=profile)

# views.py - AFTER (working)

form = VolunteerProfileForm(request.POST, request.FILES, instance=profile)

<!-- template - BEFORE (not working) -->

<form method="post">

<!-- template - AFTER (working) -->

<form method="post" enctype="multipart/form-data">

**Result:** Image uploads now work perfectly

**Challenge 2: Pagination Breaking Search Filters**

**Problem:**  
When users searched for events and then clicked pagination (page 2, 3, etc.), the search query and category filter were lost, showing all events again.

**Root Cause:**  
Pagination URLs didn't preserve the GET parameters (query and category) from the search form.

**Solution (Niket):**

<!-- BEFORE (lost filters) -->

<a href="?page=2">Page 2</a>

<!-- AFTER (preserves filters) -->

<a href="?page=2{% if request.GET.query %}&query={{ request.GET.query }}{% endif %}{% if request.GET.category %}&category={{ request.GET.category }}{% endif %}">

Page 2

</a>

**Result:** Search filters now persist across pages

**Challenge 3: Dashboard Too Large and Overwhelming**

**Problem:**  
Initial volunteer dashboard was 2500px tall with large Chart.js charts, making it difficult to see all information at once. Users had to scroll excessively.

**Root Cause:**  
Too much vertical space used by:

* Large profile picture (120px)
* Stacked stat cards
* Full-size Chart.js activity chart
* Long event list (showing all)
* 2×4 badge grid

**Solution (Niket):**

1. Reduced profile picture: 120px → 45px
2. Made stats inline (row) instead of stacked
3. **Removed Chart.js charts entirely**
4. Limited event table to 5 rows with scrolling
5. Changed badges: 2×4 → 2×2 grid
6. Put impact and leaderboard side-by-side

**Result:**  
Volunteer dashboard: 2500px → 600px (76% reduction!)  
Organizer dashboard: 650px → 550px  
Much better user experience

**Challenge 4: Email Sending Failed with Gmail**

**Problem:**  
Contact form and registration emails weren't sending. Getting "Authentication failed" errors from Gmail.

**Root Cause:**  
Gmail blocks "less secure app access" and regular password logins from applications.

**Solution (Team):**

# settings.py - Use Gmail App Password

EMAIL\_HOST\_USER = 'niketbhatt28@gmail.com'

EMAIL\_HOST\_PASSWORD = 'vbhq enoi mqgx dspg' # App Password (not regular password!)

Steps taken:

1. Go to Google Account → Security
2. Enable 2-Factor Authentication
3. Generate App Password (16-character code)
4. Use App Password in Django settings

**Result:** Emails now send successfully

**Challenge 5: Capacity Overregistration**

**Problem:**  
Events were accepting more registrations than their capacity, causing overbooking.

**Root Cause:**  
No check for confirmed registrations before allowing new signups.

**Solution (Neela):**

# Check capacity before registering

confirmed\_count = event.registrations.filter(status='confirmed').count()

if confirmed\_count < event.capacity:

status = 'confirmed'

message = 'Successfully registered!'

else:

status = 'waitlist'

message = 'Added to waitlist.'

**Result:** ✅ Capacity is now enforced, waitlist works properly

**9.2 Deployment Challenges**

**Challenge 6: Static Files Not Loading After Deployment**

**Problem:**  
After deploying to PythonAnywhere, CSS and images weren't showing.

**Root Cause:**  
Static files not collected and paths not configured properly.

**Solution (Niket):**

# Collect static files

python manage.py collectstatic --noinput

# settings.py

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

STATIC\_URL = '/static/'

# PythonAnywhere - Set static file mappings

# URL: /static/

# Directory: /home/username/GreenEvents/staticfiles

**Result:** All static files now load correctly

**Challenge 7: Database Migration to Production**

**Problem:**  
How to transfer development database (with 130 users, 60 events, 300+ registrations) to production without pushing db.sqlite3 file.

**Root Cause:**  
Database files shouldn't be in version control (in .gitignore).

**Solution (Team):**

# Export to JSON fixtures

python manage.py dumpdata auth.User > fixtures/users.json

python manage.py dumpdata events.Event > fixtures/events.json

# ... etc

# On production:

python manage.py migrate # Create tables

python manage.py loaddata fixtures/\*.json # Load data

**Result:** Portable database in JSON format

**9.3 Collaboration Challenges**

**Challenge 8: Merge Conflicts in Git**

**Problem:**  
Multiple team members editing the same files (views.py, models.py) caused merge conflicts.

**Solution:**

* Regular communication about who's working on what
* Frequent commits with descriptive messages
* Pull latest changes before starting work
* Use separate functions for each feature
* Code reviews before merging

**Result:** Minimal conflicts, smooth collaboration

**Challenge 9: Task Distribution Fairness**

**Problem:**  
Ensuring equal contribution from all team members.

**Solution:**

* Clear task distribution table from start
* Regular progress meetings
* Git commit tracking
* Individual code sections
* Each member has distinct features

**Result:** Equal 20% contribution from each member

**10. TESTING**

**10.1 Manual Testing**

**Test Case 1: User Registration**

**Tested by:** Shaurya  
**Steps:**

1. Go to signup page
2. Fill in volunteer form with image
3. Submit

**Expected:** Account created, redirected to home, logged in  
**Actual:** Passed

**Test Case 2: Event Creation**

**Tested by:** Zin  
**Steps:**

1. Login as organizer
2. Click "Create Event"
3. Fill form with all fields
4. Submit

**Expected:** Event created, redirected to event detail  
**Actual:** Passed

**Test Case 3: Event Registration**

**Tested by:** Neela  
**Steps:**

1. Login as volunteer
2. Browse events
3. Click event
4. Click "Register"

**Expected:** Registration confirmed, email sent  
**Actual:** Passed

**Test Case 4: Capacity Limit**

**Tested by:** Neela  
**Steps:**

1. Create event with capacity 2
2. Register volunteer 1 (confirmed)
3. Register volunteer 2 (confirmed)
4. Register volunteer 3

**Expected:** Volunteer 3 goes to waitlist  
**Actual:** Passed

**Test Case 5: Search Functionality**

**Tested by:** Niket  
**Steps:**

1. Go to home page
2. Search "tree"
3. Check results

**Expected:** Only events with "tree" in title/description shown  
**Actual:** Passed

**Test Case 6: Pagination**

**Tested by:** Niket  
**Steps:**

1. Home page with 61 events
2. Check page 1 (should show 20)
3. Click page 2
4. Check events 21-40 shown

**Expected:** 20 events per page, navigation works  
**Actual:** Passed

**Test Case 7: Pagination with Search**

**Tested by:** Niket  
**Steps:**

1. Search "beach"
2. See paginated results
3. Click page 2

**Expected:** Search query preserved on page 2  
**Actual:** Passed

**Test Case 8: Profile Image Upload**

**Tested by:** Shaurya  
**Steps:**

1. Login as volunteer
2. Edit profile
3. Upload image
4. Save

**Expected:** Image displays on dashboard  
**Actual:** Passed

**Test Case 9: Email Notifications**

**Tested by:** Neela  
**Steps:**

1. Register for event
2. Check email

**Expected:** Confirmation email received  
**Actual:** Passed

**Test Case 10: Google OAuth**

**Tested by:** Shaurya  
**Steps:**

1. Click "Sign in with Google"
2. Choose Google account
3. Complete profile

**Expected:** Account created via Google  
**Actual:** Passed

**10.2 Browser Testing**

**Tested On:**

* Chrome 120 (Windows)
* Firefox 121 (Windows)
* Edge 120 (Windows)
* Safari 17 (Mac)
* Chrome Mobile (Android)

**10.3 Responsive Testing**

**Tested Resolutions:**

* 1920×1080 (Desktop)
* 1366×768 (Laptop)

**10.4 Test Results Summary**

| **Feature** | **Test Cases** | **Passed** | **Failed** |
| --- | --- | --- | --- |
| Authentication | 5 | 5 | 0 |
| Event CRUD | 8 | 8 | 0 |
| Registration | 6 | 6 | 0 |
| Search & Filter | 4 | 4 | 0 |
| Pagination | 3 | 3 | 0 |
| Email | 3 | 3 | 0 |
| File Upload | 4 | 4 | 0 |
| Dashboard | 5 | 5 | 0 |
| **TOTAL** | **38** | **38** | **0** |

**Success Rate:** 100%

**11. CONCLUSION**

**11.1 Project Success**

GreenEvents successfully delivers a comprehensive platform that:

**Connects volunteers with organizations** - 130 active users, 60 events  
**Facilitates event management** - Full CRUD operations, registration system  
**Tracks environmental impact** - Trees, CO2, waste, hours  
**Provides analytics** - Real-time dashboards for both user types  
**Offers seamless UX** - Responsive design, intuitive navigation  
**Demonstrates technical proficiency** - Django, authentication, file uploads, email, deployment

**11.2 Learning Outcomes**

**Technical Skills Gained:**

* Full-stack web development with Django
* Database design and ORM usage
* User authentication (Django + OAuth)
* File upload handling
* Email integration (SMTP)
* Search and pagination
* Session management
* Version control (Git/GitHub)
* Deployment and hosting
* JSON fixtures and data migration

**Soft Skills Developed:**

* Team collaboration and communication
* Task distribution and management
* Problem-solving under deadlines
* Code review practices
* Documentation writing
* Project planning
* Time management

**11.3 Individual Achievements**

**Zin:**

* Mastered Django models and relationships
* Implemented complete CRUD functionality
* Created professional form validations
* Generated comprehensive fixtures

**Neela:**

* Built complex registration logic
* Implemented email notification system
* Managed capacity and waitlist features
* Handled re-registration scenarios

**Niket:**

* Implemented advanced search with Q objects
* Built pagination with parameter preservation
* Redesigned dashboards for better UX
* Optimized database queries

**Munna:**

* Managed user sessions effectively
* Implemented activity tracking
* Created authentication flows
* Built user history system

**Shaurya:**

* Handled file uploads properly
* Integrated Google OAuth
* Designed responsive UI
* Created professional styling
* Built static pages

**11.4 Impact**

**Environmental:**

* Platform enables measurable environmental action
* Facilitates community engagement
* Tracks collective impact (trees, CO2, waste)

**Educational:**

* Demonstrates real-world application development
* Portfolio-worthy project for all members
* Professional coding practices

**Social:**

* Brings communities together
* Promotes sustainable living
* Creates networking opportunities for volunteers

**11.5 Future Enhancements**

**Phase 1: Social Features**

* User-to-user messaging
* Event comments and reviews
* Social media sharing
* Friend/following system

**Phase 2: Mobile App**

* React Native mobile application
* Push notifications
* QR code check-in

**Phase 3: AI Integration**

* Event recommendations (ML-based)
* Smart matching (volunteers to events)
* Predictive analytics

**Phase 4: Gamification**

* More achievement badges
* Monthly/yearly leaderboards
* Volunteer challenges
* Reward system

**Phase 5: Cleaner Modularization**

* Django Apps Separation
* Easier Maintenance based Apps Integration

**11.6 Final Thoughts**

Building GreenEvents has been an enriching experience that combined technical challenges with social purpose. The project demonstrates proficiency in Django web development while addressing a real-world need for environmental community building.

The collaborative nature of the project enhanced our teamwork skills, and the structured task distribution ensured everyone contributed meaningfully to the codebase. The use of version control, fixtures, and proper deployment practices prepares us for real-world software development careers.

We're proud to present GreenEvents as a **functional, scalable, and impactful web application** that makes a difference.

**PROJECT STATISTICS**

* **Development Time:** 11 weeks
* **Total Lines of Code:** 3,000+
* **Total Commits:** 80+
* **Team Members:** 5
* **Features Implemented:** 20+
* **Test Cases:** 38 (100% passed)
* **Database Models:** 6
* **Views:** 25+
* **Templates:** 30+
* **API Integrations:** 2 (Gmail SMTP, Google OAuth)

**APPENDIX**

**A. Test Accounts**

**Volunteers:**

* volunteer\_1 through volunteer\_100
* Password: password123

**Organizers:**

* organizer\_1 through organizer\_30
* Password: password123

**B. Important Links**

* **GitHub:** https://github.com/IShaurya/GreenEvents
* **Live Site:** https://[username].pythonanywhere.com
* **Documentation:** README.md in repository

**C. File Structure**

GreenEvents/

├── greenevents/ # Project settings

├── events/ # Main app

│ ├── models.py # All 6 models

│ ├── views.py # 25+ views

│ ├── forms.py # 8 forms

│ ├── urls.py # URL routing

│ ├── admin.py # Admin config

│ └── templates/ # 30+ templates

├── fixtures/ # JSON data

│ ├── users.json # 130 users

│ ├── events.json # 60 events

│ └── ... # Other fixtures

├── static/ # CSS, JS

├── media/ # User uploads

├── manage.py # Django CLI

└── requirements.txt # Dependencies

**D. Technologies Summary**

| **Category** | **Technology** | **Version** |
| --- | --- | --- |
| Framework | Django | 5.2.7 |
| Language | Python | 3.13.2 |
| Database | SQLite | 3.x |
| Frontend | Bootstrap | 5.3 |
| Auth | django-allauth | 65.3.0 |
| Images | Pillow | 11.0.0 |
| Server | Gunicorn | 21.2.0 |