Final Project

Non-Profit Environment Conservation and Animal Protection Website
Final Report

Priya Ambaldhage

(GitHub link: https://github.com/Priya2825/ecoguardian.git)
(To access project codes please select the branch to main instead of master)

Table of Contents

Introduction:

For this project I will be doing a nonprofit website from the advanced web development module. In recent times there are significant challenges in mobilizing and sustaining community efforts towards conservation goals.

In response to these challenges, this project aims to develop a comprehensive web application dedicated to environmental conservation and animal protection. The application is designed to facilitate community engagement by offering features such as volunteer and event management, a donation platform, educational resources, environmental news updates, and recycling options. By integrating these functionalities into a user-friendly interface, the project seeks to empower individuals to actively participate in conservation efforts and make a tangible difference.

The primary objectives of this project are to raise awareness about environmental and animal protection issues, provide a platform for community involvement, and support sustainable practices. By integrating features such as volunteer opportunities, event registration, donation options, and access to educational materials, the application will serve as a comprehensive central hub for various conservation activities.

This report will provide a detailed overview of the project, beginning with a literature review that examines existing initiatives and the technologies or languages relevant to environmental conservation. Following this, the report will outline the methodology employed in developing the application, incorporating an in-depth discussion of the implementation process and the features integrated into the platform. The review will also cover the utilization of technologies like the Django web framework for robust and scalable web application development, MySQL for efficient data management, and Adobe XD for creating high-fidelity wireframes and prototypes. Finally, the report will explore the expected outcomes and potential impact of the project on conservation efforts.

By harnessing the power of advanced technologies and fostering community involvement, this project aspires to significantly contribute to the global movement for environmental sustainability and animal welfare. Through a meticulously designed and user-friendly web application, it aims to create a positive and lasting impact on our planet, encouraging more individuals to engage in and support conservation efforts actively.

In conclusion, the website is designed for a diverse audience dedicated to environmental conservation and animal protection. It targets volunteers who are eager to engage in conservation activities and students seeking community service opportunities. Donors, whether individuals or organizations, will find the donation platform particularly beneficial for supporting conservation projects. Enthusiasts of environmental and animal protection will appreciate the comprehensive educational resources, informative content, and courses available on the site. Community members interested in local initiatives, as well as families and groups looking for community events, will discover numerous ways to get involved. Additionally, Non-Governmental Organizations (NGOs) focused on conservation can use the platform to recruit volunteers and promote their initiatives, thereby enhancing their impact. By catering to the needs of this diverse audience, the website aims to build a vibrant community committed to environmental sustainability and animal welfare, fostering active participation and support from all sectors of society.

Motivation:

The non-profit website dedicated to environmental conservation and animal protection aims to make a significant impact by providing a user-friendly platform that offers versatile support for donations and volunteering.

This project is motivated by the need to raise awareness about environmental issues and animal welfare through informative content and engaging stories, inspiring visitors to advocate for conservation. By

offering a wide range of educational resources and courses, the website empowers users with the knowledge needed to make informed decisions and become passionate advocates for the environment. This educational component is crucial as it helps bridge the gap between awareness and action, ensuring users are well-equipped to participate in conservation efforts effectively.

Facilitating community engagement through volunteer opportunities and events will encourage collective participation in conservation activities, fostering a sense of community responsibility. By providing a comprehensive calendar of events and volunteer opportunities, the platform makes it easy for users to find and participate in activities that align with their interests and availability. The flexible donation platform supports various conservation projects and initiatives, allowing individuals to contribute in ways that suit their capacity, whether through one-time donations, recurring contributions, or even in-kind donations of goods and services.

Additionally, the website will offer practical tips on recycling and sustainable practices, along with features like recycling options and reward systems. These elements will promote eco-friendly habits among users, encouraging them to adopt sustainable practices in their daily lives. The inclusion of a reward system for recycling not only incentivizes participation but also provides a tangible way for users to see the impact of their actions.

The platform will also empower external NGOs by providing them a space to recruit volunteers and promote their initiatives, enhancing the overall impact of conservation efforts. By creating a centralized hub for various organizations, the website fosters collaboration and amplifies the reach and effectiveness of individual initiatives. This collaborative approach ensures that resources are used efficiently and that conservation efforts are aligned towards common goals.

By addressing these key areas, the project seeks to create a comprehensive and impactful platform that not only raises awareness but also drives meaningful action towards environmental sustainability and animal welfare. Ultimately, this project aims to contribute to a sustainable future for all, ensuring that current and future generations can enjoy a healthy and thriving natural world.

(862 words)

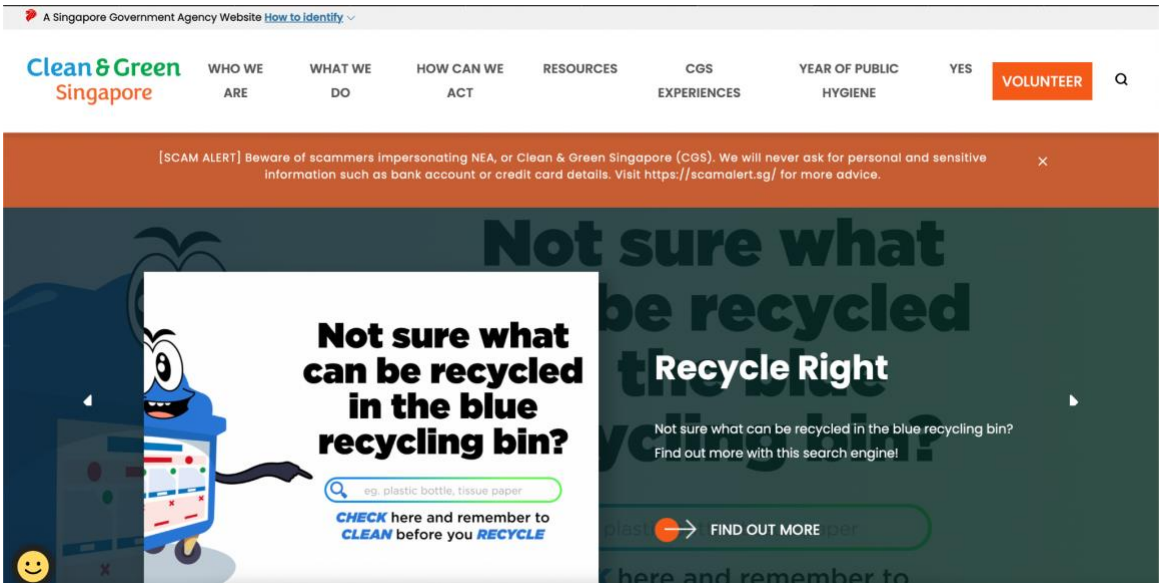
Literature Review:

This literature review delves into the landscape of existing projects, techniques, and research studies pertinent to the development of a web application focused on environmental conservation and animal protection. By meticulously examining examples of similar initiatives, the techniques and methods intended for implementation, and research studies that demonstrate effectiveness, this review seeks to shed light on the current state of the field. The objective is to uncover opportunities for innovation and enhancement in conservation efforts through the creation of a multifaceted web platform.

Furthermore, this comprehensive analysis of existing literature is designed to inform the development process of a user-friendly and impactful web application. The goal is to empower users by providing them with the tools and resources needed to actively participate in conservation activities, thereby fostering a collective effort toward environmental sustainability and animal welfare. Ultimately, this review aspires to contribute to the broader discourse on conservation by identifying best practices, highlighting gaps, and suggesting avenues for future research and development. Through this endeavor, the aim is to create a platform that not only informs but also inspires action, enabling users to make a tangible positive impact on a global scale.

Examples of Similar Projects:

a. Clean and Green Singapore Website [i]

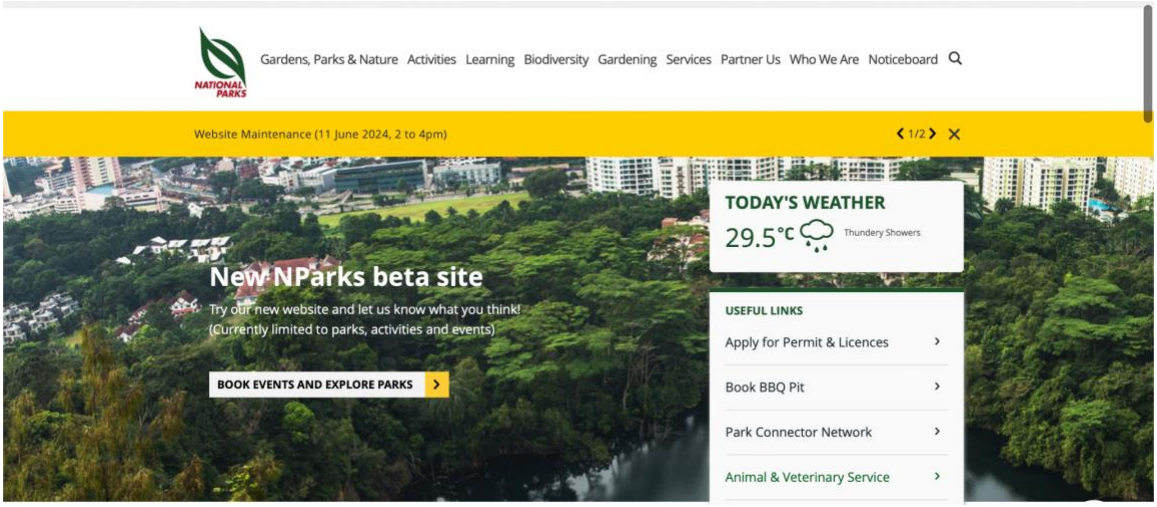


- A website that effectively highlights the importance of volunteering and demonstrates its positive impact is Clean and Green Singapore. This platform excels with its beginner-friendly interface, offering accessible features such as recognition for awards and a variety of volunteering options, thereby empowering individuals to actively participate in environmental initiatives. Notably, the site includes a dedicated "Volunteer" tab prominently positioned at the corner of the webpage in a bold, colorful box, making it easy for volunteers to quickly access their specific resources. Furthermore, the navigation bar is thoughtfully designed, featuring sections for events, awards, volunteer resources, and more. This organization of content not only enhances user experience but also facilitates efficient navigation, ensuring users can find relevant information with ease.
- By studying and drawing inspiration from these elements, the development of my website will aim to integrate similar user-centric design principles, promoting better engagement and usability.

b. Singapore Zoo[ii]

- Their website offers functionalities that align closely with my vision, providing essential support for donations and volunteering opportunities as conservation ambassadors. Additionally, they have thoughtfully integrated valuable study materials, which serve as an excellent resource for users interested in deepening their knowledge. However, I perceive several areas where enhancements can be made to improve user experience and engagement.
- One notable aspect where their platform differs is in the handling of donations. On their site, donations are mandatory even for volunteers, which may discourage some users from participating. In contrast, my vision includes creating distinct features that allow users to choose between donating and volunteering without feeling obligated to do both.
- Furthermore, the layout for volunteering opportunities on their website could be more user-friendly. Currently, users must scroll down to view different activities, which can be time-consuming. Despite this layout challenge, they have categorized these opportunities into three distinct sections: Advocate for Wildlife, Animal Care, and Veterinary and Research. This structured approach helps users quickly identify areas of interest and find relevant opportunities.
- To improve the overall user experience, my plan involves creating a more intuitive and streamlined interface for navigating volunteering opportunities. By implementing a dynamic and interactive design, users will be able to browse and select activities easily, without excessive scrolling. Additionally, separating the donation and volunteering features will offer greater flexibility and cater to a broader audience, thereby encouraging more users to engage with the platform.

c. National Park Singapore [iii]

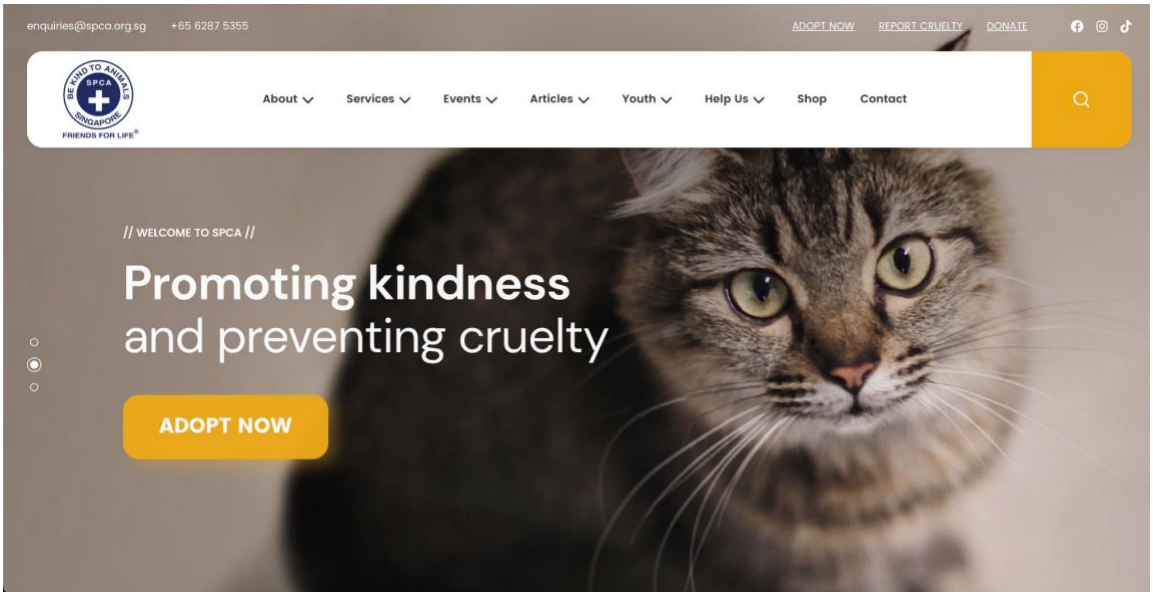


- The National Park Singapore website stands out for its exceptional user-friendliness and intuitive design. The website's navigation bar clearly labels and organizes all key sections such as events and workshops, tours, volunteering opportunities, sponsoring options, awards, internships, information about the organization, learning resources, and park guidelines. This clear and accessible layout ensures that visitors can effortlessly locate the information they need, enhancing their overall experience.
- The homepage is particularly effective in engaging users by prominently featuring options for booking events and exploring parks. These prominent features are strategically designed to draw users' attention and encourage active participation in the park's offerings. Additionally, the homepage includes a well-placed "useful links" corner, which provides quick access to important resources, ensuring that users have all the necessary information readily available.
- This thoughtful layout not only enhances the user experience by making navigation seamless and intuitive but also promotes greater involvement in park activities and conservation efforts. By prioritizing user needs and accessibility, the website effectively fosters a deeper connection between visitors and the park's mission. The design choices, such as clear labeling, prominent

call-to-action buttons, and accessible resources, collectively contribute to a more engaging and user-centric platform.

- In developing my project, I aim to emulate these principles by creating a similarly organized and user-friendly interface. This approach will ensure that users can easily navigate the site, access important information, and engage with various conservation activities. By integrating such user-centric design elements, the project will effectively support environmental conservation and animal protection initiatives, fostering a greater sense of community involvement and awareness.

d. Society of the Prevention of Cruelty to Animals (SPCA)[iv]



- The website is meticulously designed to captivate and engage its users. Positioned just below the navigation bar, there are three prominent links: "Adopt Now," "Report Cruelty," and "Donate." These links are strategically placed to be highly visible, utilizing white font against a dark gray banner background, with underlines to emphasize their significance and draw immediate attention to critical issues. The homepage layout is thoughtfully sectioned into four distinct areas: a banner image accompanied by a search bar, hovering containers that highlight key information, a "2023 at a Glance" section displaying the number of animals rescued and treated, and a blog section featuring upcoming events. This structured approach ensures that important information is easily accessible and prominently displayed.
- Additionally, the website's search functionality is uniquely designed to enhance user focus. When the search bar is engaged, the background darkens, leaving only the search bar illuminated. This design choice effectively directs the user's attention solely to their search query, improving the overall user experience by making navigation intuitive and engaging. By incorporating such user-centric design principles, the website ensures that users can quickly and easily access vital information and resources. This approach not only facilitates better user engagement but also highlights the importance of the site's mission in a visually compelling manner. In developing my project, I aim to integrate similar design elements to create a seamless, user-friendly interface that effectively supports environmental conservation and animal protection initiatives.

Techniques and Methods Planned for Use:

a. Django Web Framework[v]

- From my perspective, the Django Web Framework is pivotal for this project. Known for its rapid development capabilities and adherence to a pragmatic design philosophy, Django offers a sturdy foundation with features such as authentication, URL routing, and database ORM. By leveraging

Django's MVC architecture and extensive community support, our objective is to streamline development and deliver a user-centric web application dedicated to environmental conservation and animal protection.

- Moreover, in addition to prioritizing accessibility, incorporating responsive design practices to ensure optimal viewing across various devices. Nonetheless, achieving consistent responsiveness across diverse browsers and devices presents challenges, necessitating thorough testing and troubleshooting.
- To uphold code quality and reliability, employing rigorous testing methodologies and debugging tools. While essential for ensuring a seamless user experience, these practices may introduce challenges related to time constraints and the need for continuous integration and deployment.

b. Adobe XD

- From my perspective, Adobe XD is perfect for crafting both wireframes and high-fidelity prototypes. Renowned for its intuitive interface and robust features tailored for UI/UX design, Adobe XD offers an ideal platform for this purpose. By harnessing the capabilities of Adobe XD, I envisage meticulously designing detailed wireframes and visually captivating prototypes. These elements will play a crucial role in realizing our project's overarching mission of advocating for environmental conservation and animal protection.

c. SQLite

- SQLite, a lightweight and serverless database engine, is perfectly suited for our non-profit environmental conservation and animal protection website. Its single-file storage system simplifies database management, allowing for seamless integration and easy backups

(1457 words)

Design:

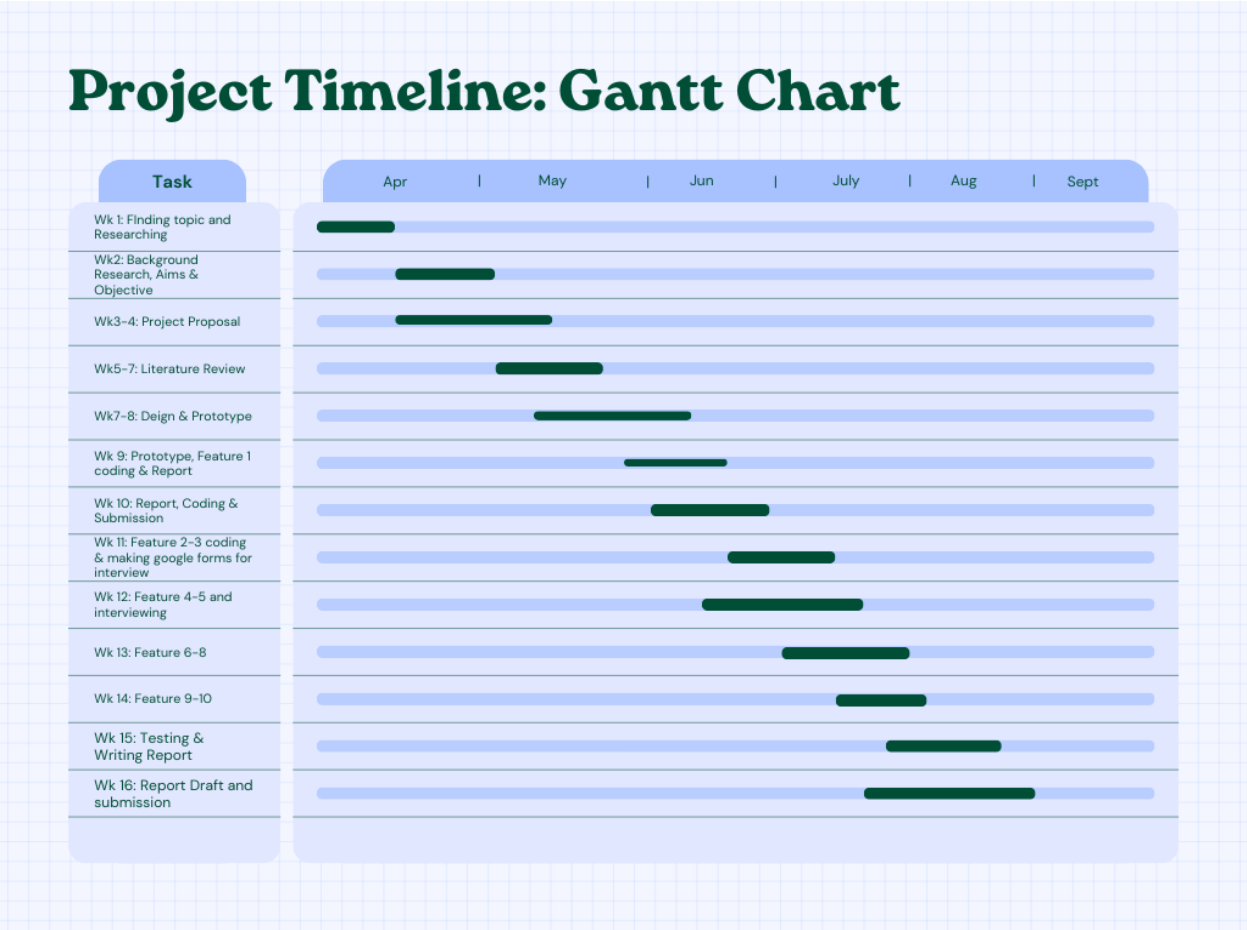
Domain and users:

Project domain is environmental conservation and animal protection.

Users:

1. Volunteers: Individuals interested in contributing their time and effort to support environmental and wildlife conservation activities. These users need access to features like event sign-up, participation tracking, and information on upcoming volunteer opportunities.
2. Donors: People who want to financially support conservation initiatives. These users require a seamless, secure donation process and might also want to receive updates on how their contributions are making an impact.
3. General Public: Individuals seeking information on environmental and wildlife conservation. They might use the website to learn more about conservation efforts, stay updated on news, or find ways to get involved.

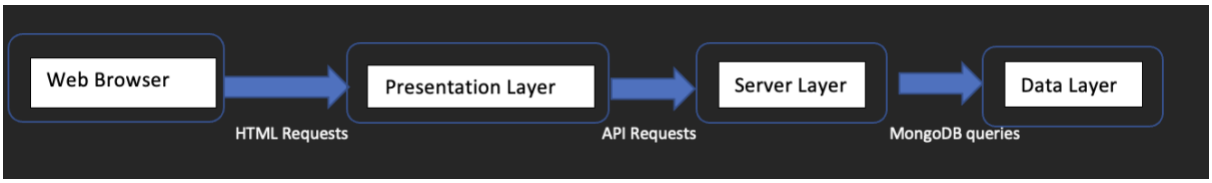
Gantt Chart:


Project Timeline: Gantt Chart
Task
Apr
May
Jun
July
Aug
Sept
Wk 1: FInding topic and Researching
Wk2: Background Research, Aims & Objective
Wk3-4: Project Proposal
Wk5-7: Literature Review
Wk7-8: Deign & Prototype
Wk 9: Prototype, Feature 1 coding & Report
Wk 10: Report, Coding & Submission
Wk 11: Feature 2-3 coding & making google forms for interview
Wk 12: Feature 4-5 and interviewing
Wk 13: Feature 6-8
Wk 14: Feature 9-10
Wk 15: Testing & Writing Report
Wk 16: Report Draft and submission

	Task	Type of Assignment	Start	End	Dur	%	2024 Apr	May	Jun	Jul	Aug	Sep
	Project		1/4/24	9/9/24	116							
1	Wk 1: Finding topic and Researching Elements	Researching	1/4/24	1/4/24	1							
2	Wk 2: Background Research, Aims & Objectives	Introduction Section	19/4/24	19/4/24	1							
3	Wk 3-4: Project Proposal	Google Slides	20/4/24	26/4/24	5							
4	Wk 5-6: Literature Review	Report Introduction	27/4/24	7/5/24	7							
5	Wk 7-8: Design & Prototype	Low-Fidelity Wireframes	8/5/24	14/5/24	5							
6	Wk 9: Prototype, Feature 1 Coding & Report Writing	Coding of User Registration and Authentication	14/5/24	2/6/24	14							
7	Wk 10: Report, Feature 2 Coding & Submission	Coding of Volunteer Opportunities	3/6/24	15/6/24	10							
8	Wk 11: Feature 3 Coding	Coding of Event Registration and Management	19/6/24	23/6/24	3							
9	Wk 11: Feature 4 Coding	Coding of Donation Platform	23/6/24	28/6/24	5							
10	Wk 12: Feature 5 Coding	Coding of Email Notifications	28/6/24	28/6/24	1							
11	Wk 12: Feature 6 Coding	Coding of Add to Calendar	28/6/24	3/7/24	4							
12	Wk 13: Feature 7 Coding	Coding of Report Page	4/7/24	12/7/24	7							
13	Wk 14: Creating Google form and Gathering responses	Google Forms	13/7/24	18/7/24	4							
14	Wk 15: Writng Draft Report	Draft Report Writing	18/7/24	22/7/24	3							
15	Wk 16: Improving existing layout for all pages		24/7/24	31/7/24	6							
16	Wk 17: Feature 8 Coding	Coding of Adopt Now Page	2/8/24	10/8/24	6							
17	Wk 18: Feature 9 Coding	Coding of Report Now Page	11/8/24	16/8/24	5							
18	Wk 19: Feature 10 Coding	Coding of Recycle Page	26/8/24	31/8/24	5							
19	Testing of Features / Documentation	Testing / User Testing	1/9/24	4/9/24	3							
20	Completing Documentation and Submission		5/9/24	9/9/24	3							

This Gantt chart outlines the project timeline for developing a non-profit website focused on environmental conservation and animal protection using Django and MySQL. The chart details the tasks and milestones over a 20-week period.

Architectural Design:



- The above diagram depicts a 3-tier architecture drawn using Microsoft Word. In this architecture, the web browser serves as the user interface, enabling users to browse the environmental conservation website. Users can view and interact with features related to volunteering opportunities, donations, and other site functionalities.

- The presentation layer, or client tier, sends HTTP requests to the server tier to retrieve details about volunteering and recycling opportunities. This layer also handles client requests and is primarily coded in HTML, CSS, Python, and Django.

- The server layer, or server tier, processes these client requests and manages server-side operations to handle application logic. This layer ensures that the necessary data and functionality are provided to the client tier.

- Finally, the database layer employs MySQL to store all information related to volunteers, organizations, and administrators. It performs CRUD operations (CREATE, READ, UPDATE, DELETE) to maintain and manage the database effectively.

Methodology:

The development of this website was guided by an Agile methodology, tailored to accommodate the specific needs of building a non-profit platform focused on environmental conservation and animal protection. This approach was structured around five key phases: Plan, Design, Develop, Test, and Deploy, with constant collaboration and iterative development to ensure that the project could adapt swiftly to any changes.

Planning Phase:
The planning phase began with setting clear goals that were specific to the mission of the website: facilitating community engagement, managing volunteers, and supporting fundraising efforts for conservation. I conducted a needs assessment by reviewing similar non-profit websites and engaging with potential stakeholders to define the project's scope. This phase also involved selecting Django as the primary framework due to its robustness and suitability for building secure, scalable web applications.

Design Phase:
In the design phase, I created detailed wireframes and a database schema that were specifically tailored to the functionalities required by the website. For example, the volunteer management system needed to be intuitive for both users and administrators, so I designed the user interface with simplicity and ease of navigation in mind. The database schema was carefully structured to efficiently handle user data, event registrations, and donation records, ensuring that the backend could support the anticipated user load without performance issues.

Development Phase:
During the development phase, I built the website's core features in iterative sprints, focusing on one functionality at a time. For instance, the user authentication system was implemented first to ensure that secure access controls were in place from the outset. Following that, I developed the volunteer management module, which included features such as event sign-up, participation tracking, and automated email reminders. Each feature was developed with the specific needs of a non-profit organization in mind, incorporating feedback loops to refine the functionality based on user testing.

Testing Phase:
Rigorous testing was conducted at each stage of development to ensure the highest quality and reliability. Unit tests were written for each Django model and view, ensuring that individual components performed as expected. Integration testing was particularly crucial for the donation platform, where I tested the interaction between the payment gateway and the website to guarantee secure and smooth transaction

processing. Additionally, I conducted user acceptance testing with a small group of potential users, gathering feedback on the website's usability and making necessary adjustments to improve the user experience.

Features:

Our non-profit website aims to leverage modern technology to enhance engagement, streamline operations, and maximize impact for our cause. Developed using Django and MySQL, our platform incorporates the following key features:

Functional Features:

1. User Registration and Authentication
    - Users can create accounts and log in securely.
2. Volunteer Opportunities
    - Browse and search for volunteer opportunities.
    - Sign up for volunteer activities.
    - Volunteer dashboard to track participation and upcoming events.
3. Event Registration and Management
    - View and register for upcoming events.
    - Event organizers can create and manage events.
    - Calendar integration to view event schedules.
4. Donation Platform
    - Make one-time or recurring donations.
    - Track donation history.
5. Educational Resources and Blogs
    - Access to articles, videos, and courses on environmental conservation and animal protection
6. Recycling Tips and Options
    - Information on how to recycle various materials.
    - Options to arrange for pick-up of recyclable items.
    - Reward system for recycling participation.
7. NGO Collaboration
    - Platform for external NGOs to post volunteer opportunities.
    - Tools for NGOs to manage volunteer recruitment and event promotion.
8. Pop-Up Chat Feature
    - Live chat support for user inquiries and assistance.
9. Email Notifications
    - Alerts for upcoming events and volunteer opportunities.
    - Newsletters and updates on conservation projects.
10. Admin Profile

Non-Function Features:

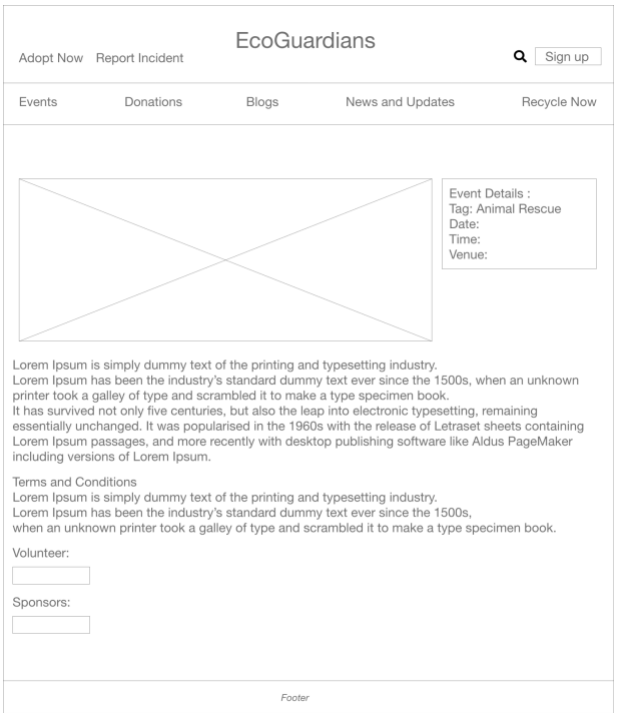
1. Usability
    - Intuitive and user-friendly interface.
2. Security
    - Secure user authentication and authorization.
    - Data encryption for sensitive information.
3. Compatibility
    - Responsive design for mobile and tablet devices

UI Design:

Prototype:

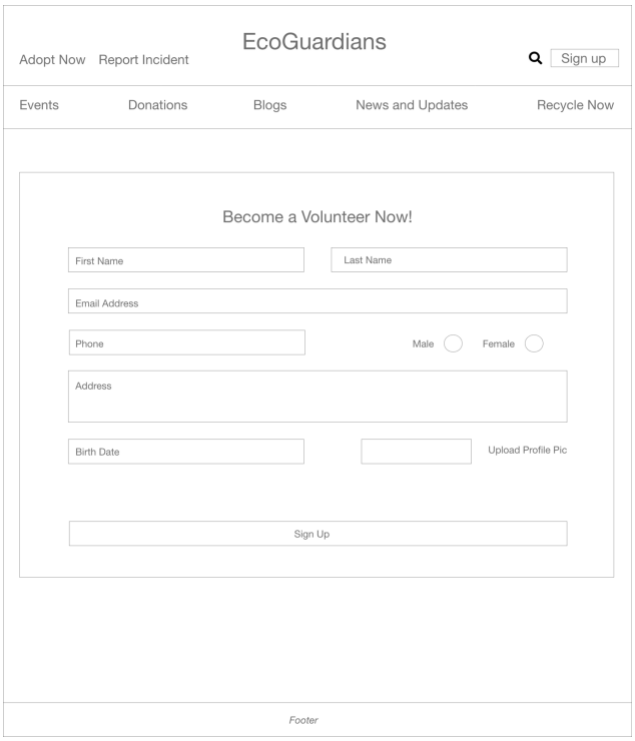
Software used: AdobeXD
Home Page

- Functionality included: search, User Registration and Authentication



Event Page

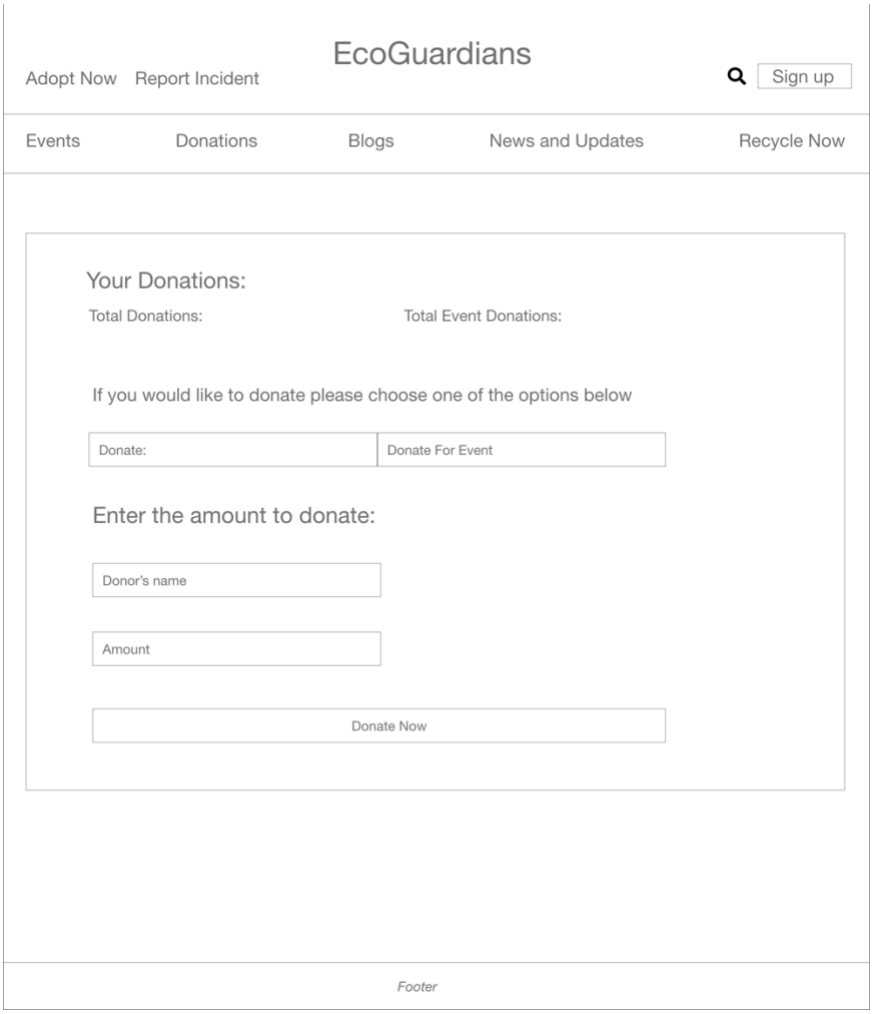
- View and register for upcoming events.



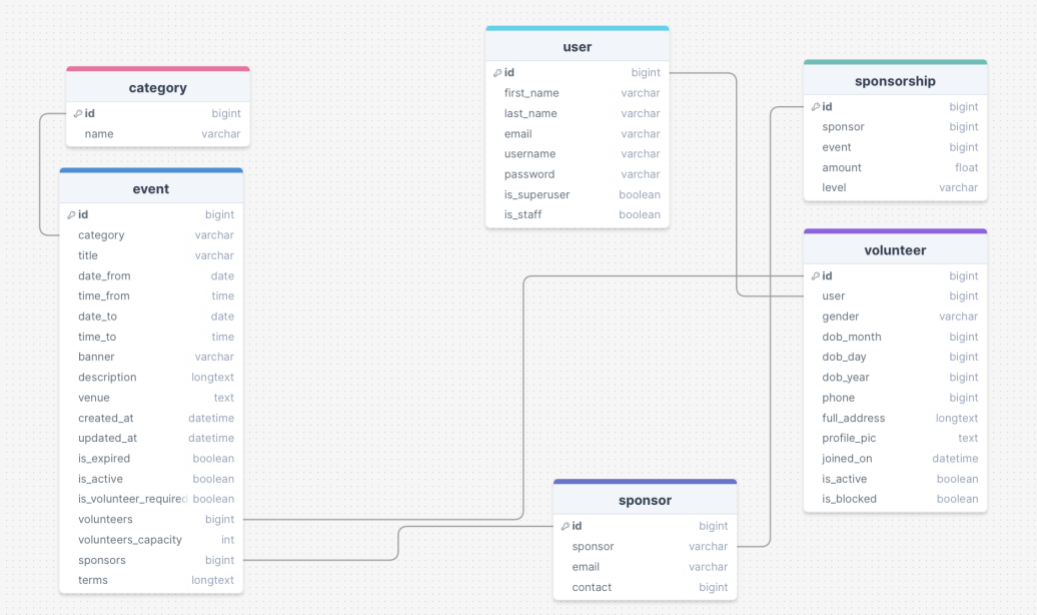
Register Page

- Event Registration and Management, creating profile for donation, recycling and volunteering opportunities

Donation Page:



ER Diagram:



The provided Entity-Relationship (ER) diagram illustrates the structured design of a database intended for an environmental conservation and animal protection web application. This diagram showcases six primary entities: User, Category, Event, Volunteer, Sponsor, and Sponsorship, each with its respective attributes and defined relationships.

1. User: Contains user details like `id`, `first_name`, `last_name`, `email`, `username`, `password`, and roles (`is_superuser`, `is_staff`).

2. Category: Contains categories for events with attributes `id` and `name`.

3. Event: Holds information about events, such as `id`, `category`, `title`, `date_from`, `time_from`, `date_to`, `time_to`, `banner`, `description`, `venue`, `created_at`, `updated_at`, `is_expired`, `is_active`, `is_volunteer_required`, `volunteers`, `volunteers_capacity`, `sponsors`, and `terms`.

4. Volunteer: Captures volunteer details with attributes like `id`, `user`, `gender`, `dob_month`, `dob_day`, `dob_year`, `phone`, `full_address`, `profile_pic`, `joined_on`, `is_active`, and `is_blocked`.

5. Sponsor: Includes sponsor information with `id`, `sponsor`, `email`, and `contact`.

6. Sponsorship: Links sponsors to events with attributes like `id`, `sponsor`, `event`, `amount`, and `level`.

Relationships:

- User and Volunteer: A user can be linked to one volunteer profile.

- Category and Event: Each category can have multiple events.

- Event and Sponsorship: Each event can have multiple sponsors, and each sponsor can support multiple events.

This diagram provides a structured way to manage user interactions, event details, volunteer information, and sponsorships, ensuring that all data is well-organized and accessible. (1207 words)

Feature Prototype:

For Organizer App:
Models:[vi]

In Django, models define the structure of your database tables. They are Python subclasses. Here's a detailed explanation of each model in your models.py:

```python
# Create your models here.
class Category(models.Model):
    name = models.CharField(max_length=200)

    def __str__(self):
        return self.name
```

The Category model is used to categorize events based on their type

```python
class Volunteer(models.Model):
    GENDER_CHOICES = [
        ('male', 'Male'),
        ('female', 'Female'),
        ('other', 'Other')
    ]
    user = models.OneToOneField(User, on_delete=models.CASCADE, blank=True, null=True)
    gender = models.CharField(max_length=50, choices=GENDER_CHOICES, default='male')
    dob_month = models.IntegerField()
    dob_day = models.IntegerField()
    dob_year = models.IntegerField()
    phone = models.CharField(max_length=10)
    full_address = models.TextField()
    profile_pic = models.ImageField(upload_to='profile_pics/', blank=True, null=True)
    joined_on = models.BooleanField(default=True)
    is_active = models.BooleanField(default=True)
    is_blocked = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

The Volunteer model in your Django application plays a crucial role in managing and organizing volunteers who participate in your nonprofit's activities and events. It includes fields for personal details (gender, dob_month, dob_day, dob_year, phone, full_address), profile picture (profile_pic), and status (joined_on, is_active, is_blocked). It integrates with Django's user system to ensure efficient management and engagement of volunteers.

```python
class Event(models.Model):
    title = models.CharField(max_length=500)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    date_from = models.DateField()
    date_to = models.DateField()
    time_from = models.TimeField()
    time_to = models.TimeField()
    banner = models.ImageField(upload_to='event_banners/', blank=True, null=True)
    description = models.TextField()
    venue = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    is_expired = models.BooleanField(False)
    is_active = models.BooleanField(True)
    is_volunteer_required = models.BooleanField(False)
    volunteers = models.ManyToManyField(Volunteer, related_name='events')
    volunteers_capacity = models.IntegerField()
    sponsers = models.ManyToManyField(Sponser, related_name='events')
    terms = models.TextField()

    def __str__(self):
        return self.title
```

The Event model represents individual events on the nonprofit's website, with fields for title, category, dates, times, banner, description, venue, and status. It includes ManyToMany relationships with the Volunteer and Sponsor models for associating multiple volunteers and sponsors with each event.

```python
class Sponser(models.Model):
    sponser = models.CharField(max_length=100)
    email = models.EmailField()
    contact = models.CharField(max_length=8)

    def __str__(self):
        return self.sponser
```

The Sponser model stores information about organizations that sponsor events. It includes fields such as sponser, email, and contact to capture the sponsor's name, email address, and contact number respectively.

```python
class Donation(models.Model):
    donated_name = models.ForeignKey(Volunteer, on_delete=models.CASCADE)
    amount = models.FloatField()
    donated_on = models.DateTimeField(auto_now_add=True)
    if_donated_for_event = models.BooleanField(False)
    event = models.ForeignKey(Event, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.doner_name} donated {self.amount}"
```

The Donation model captures details about donations made by volunteers to the nonprofit webpage. It includes fields such as donated_name (a ForeignKey to Volunteer model), amount (the donation amount), donated_on (date and time of donation), if_donated_for_event (boolean indicating if the donation is specific to an event), and event (ForeignKey to Event model, linking the donation to a particular event).

```python
class Sponsership(models.Model):
    LEVEL_CHOICES = [
        ('gold','Gold'),
        ('silver','Silver'),
        ('bronze', 'Bronze')
    ]
    sponser = models.ForeignKey(Sponser, on_delete=models.CASCADE)
    event = models.ForeignKey(Event, on_delete= models.CASCADE)
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    level = models.CharField(max_length=100, choices=LEVEL_CHOICES)

    def __str__(self):
        return f"{self.sponser.sponser} sponserd {self.event.title}"
```

The Sponsership model records details about sponsorships for events organized by your nonprofit.It includes fields such as sponser (ForeignKey to Sponser model, identifying the sponsoring entity), event(ForeignKey to Event model, linking the sponsorship to a specific event), amount (the monetary amount of the sponsorship), and level (the sponsorship level, chosen from predefined choices).

views.py

```python
from django.shortcuts import render, redirect, get_object_or_404
from organizers.models import Event
from organizers.forms import EventForm

# Create your views here.
def home(request):
    context = {}
    all_events = Event.objects.all()
    context['all_events'] = all_events
    return render(request, 'home.html', context)

def event_details(request, event_id):
    context = {}
    event_detail = Event.objects.get(id=event_id)
    context["event_detail"] = event_detail
    return render(request, 'organizers/event_detail.html', context)

def all_events(request):
    context = {}
    all_events = Event.objects.all()
    context['all_events'] = all_events
    return render(request, 'organizers/all_events.html', context)

def create_event(request):
    if request.method == 'POST':
        form = EventForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            return redirect('all-events')
    else:
        form = EventForm()
    return render(request,'organizers/create_event.html', {'form': form})

def edit_event(request, event_id):
    event = get_object_or_404(Event, pk=event_id)
    if request.method=='POST':
        form = EventForm(request.POST, request.FILES, instance=event)
        if form.is_valid():
            form.save()
            return redirect('all-events')
    else:
        form = EventForm(instance=event)
    return render(request,'organizers/edit_event.html', {'form':form})


def delete_event(request, event_id):
    event = get_object_or_404(Event, pk=event_id)
    event.delete()
    return redirect('all-events')
```

1. Home view : The home view serves as the landing page for the website, displaying a list of all events hosted by the organization. It retrieves all Event objects from the database using Event.objects.all() and passes them to the home.html template via the context dictionary. This allows visitors to see upcoming events at a glance when they visit the homepage.

2. Event_details view: The event_details view fetches detailed information about a specific event identified by event_id. It retrieves the corresponding Event object using Event.objects.get(id=event_id) and renders this data in the organizers/event_detail.html template. This template displays comprehensive details about the event, including its title, description, dates, venue, and additional information like volunteers and sponsors associated with the event.

3. All_events view: The all_events view lists all events hosted by the organization on a dedicated page (organizers/all_events.html). It retrieves all Event objects from the database using Event.objects.all() and passes them to the template via the context dictionary. This view ensures that users can view and explore all events in a paginated format, with options to view details, edit, or delete each event directly from this centralized event listing page.

4. Create_event view: The create_event view manages the creation of new events through a form (EventForm). When the request method is POST, indicating form submission, it validates the form data (request.POST and request.FILES for file uploads) using EventForm. If the form data is valid (form.is_valid()), it saves the new Event object to the database and redirects the user to the all-events URL. If the request method is GET, it renders the organizers/create_event.html template with a blank EventForm, allowing users to input details for a new event.

5. Edit_event view: The edit_event view handles the editing of existing events identified by event_id. It retrieves the specific Event object using get_object_or_404(Event, pk=event_id) and populates an EventForm instance with the current data from the retrieved event. If the request method is POST, it processes the submitted form data (request.POST and request.FILES) to update the existing event. Upon successful validation (form.is_valid()), it saves the updated event data and redirects the user to the all-events URL. If the request method is GET, it renders the organizers/edit_event.html template pre-filled with the current event details for user modification.

6. delete _event view: The delete_event view manages the deletion of an event identified by event_id.

Urls.py

```
from django.urls import path
from organizers.views import home, event_details, all_events, create_event, edit_event, delete_event

urlpatterns = [
    path('', home, name='home'),
    path('event/<int:event_id>/', event_details, name='event_details'),
    path('all-events/', all_events, name='all-events'),
    path('create-event/', create_event, name='create_events'),
    path('edit-event/<int:event_id>/', edit_event, name='edit_events'),
    path('delete-event/<int:event_id>/', delete_event, name='delete_events'),
```

Once the API is created, the next step involves routing them through the `urls.py` file.

HTML Pages:[vii]

Base.html:[viii]

```
<body>
  <div class="container">
    <header class="border-bottom lh-1 py-3">
      <div class="row flex-nowrap justify-content-between align-items-center">
        <div class="col-4 pt-1">
          <a class="link-secondary" href="#">Adopt Now</a> 
          <a class="link-secondary" href="#">Report Incident</a>
        </div>
        <div class="col-4 text-center">
          <a class="blog-header-logo text-body-emphasis text-decoration-none fw-bold fs-1" href="/"><span class="text-success">Eco</span>Guardians</a>
        </div>
        <div class="col-4 d-flex justify-content-end align-items-center">
          <a class="link-secondary" href="#" aria-label="Search">
            <svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" fill="none" stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="mx-3" role="img" viewBox="0 0 24 24"><title>Search</
          </a>
          <a class="btn btn-sm btn-outline-secondary" href="#">Register As Volunteer</a>
        </div>
      </div>
    </header>

    <div class="nav-scroller py-1 mb-3 border-bottom">
      <nav class="nav nav-underline justify-content-between">
        <a class="nav-item nav-link link-body-emphasis active" href="{% url 'all-events' %}">Events</a>
        <a class="nav-item nav-link link-body-emphasis" href="#">Donation</a>
        <a class="nav-item nav-link link-body-emphasis" href="#">Blogs</a>
        <a class="nav-item nav-link link-body-emphasis" href="#">News and Update</a>
        <a class="nav-item nav-link link-body-emphasis" href="#">Recycle Now</a>
      </nav>
    </div>
  </div>
  <main class="container">
    {% block content %}
    {% endblock content %}
  </main>
  <footer class="py-5 text-center text-body-secondary bg-body-tertiary">
    <p>Blog template built for <a href="https://getbootstrap.com/">Bootstrap</a> by <a href="https://twitter.com/mdo">@mdo</a>.</p>
    <p class="mb-0">
      <a href="#">Back to top</a>
    </p>
  </footer>
  <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js" integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.min.js" integrity="sha384-0pUGZvbkm6XF6gxjEnlmuGrJXVbNuzT9qBBavbLwCsOGabYfZo0T0to5eqruptLy" crossorigin="anonymous"></script>
  </body>
</html>
```

The `<body>` section begins with a `<header>` that includes key links (Adopt Now, Report Incident) and the main logo (EcoGuardians) that also links to the homepage (/). It features search functionality (Search icon) and a call-to-action button (Register As Volunteer). The navigation `<nav>` below (nav-scroller) contains links (nav-item nav-link) to different site sections (Events, Donation, Blogs, News and Update, Recycle Now), ensuring easy navigation. The `<main>` section acts as a content container (container) where child templates dynamically inject specific content ({% block content %}), allowing each page to customize its content while keeping headers, navigation, and footers consistent.

Home.html

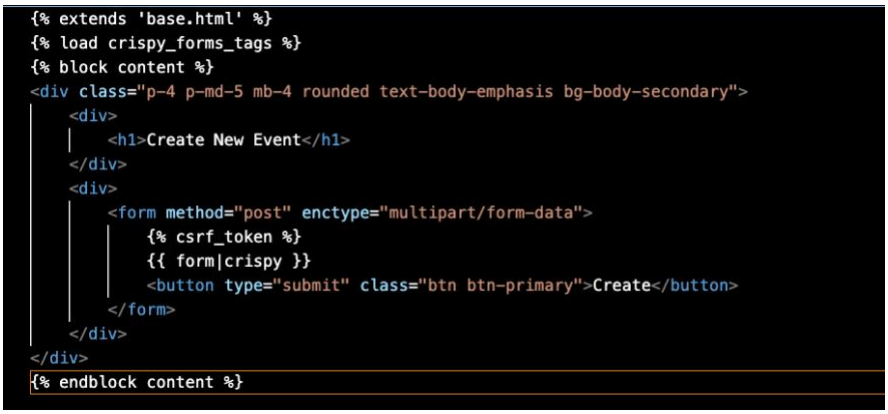
```
% extends 'base.html' %}
% block content %}
iv class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
  <div class="col-lg-6 px-0">
    <h1 class="display-4 fst-italic">Title of a longer featured blog post</h1>
    <p class="lead my-3">Multiple lines of text that form the lede, informing new readers quickly and efficiently about what's most interesting in this post's contents.</p>
    <p class="lead mb-0"><a href="#" class="text-body-emphasis fw-bold">Continue reading...</a></p>
  </div>
</div>

<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary bg-light">
  <div class="row mb-2">
    <div class="text-center">
      <h3 class="fw-bold">Current Events</h3>
    </div>
    {% for event in all_events %}
    <div class="col-sm-12 col-md-6 col-lg-6">
      <div class="card mb-3">
        <img src="{{event.banner.url}}" class="card-img-top" alt="...">
        <div class="card-body">
        <a href="{% url 'event_details' event.id %}" class="text-danger">
        <h5 class="card-title">{{event.title}}</h5>
        </a>
        <span class="badge text-bg-secondary">{{event.category}}</span>
        <p class="card-text">{{event.description|truncatechars:200}}</p>
        <p class="card-text"><small class="text-body-secondary">From : {{event.date_from}} To : {{event.date_to}}</small></p>
        <a href="{% url 'event_details' event.id %}" class="btn btn-success">Check Event</a>
        </div>
      </div>
    </div>
    {% endfor %}
  </div>
  <div class="text-end">
  <a href="{% url 'all-events' %}" class="btn btn-danger">Check all events</a>
  </div>
</div>
<div class="row g-5">…
</div>
% endblock content %}
```

The home.html template displays a list of all events (all_events) fetched from the database. Each event card (card) includes dynamic content such as event title ({{ event.title }}), banner image ({{

event.banner.url }}), and a truncated description ({{ event.description|truncatechars:200 }}). Additional details like event category ({{ event.category }}), creation date ({{ event.created_at }}), start and end date/time ({{ event.date_from }} - {{ event.time_from }}, {{ event.date_to }} - {{ event.time_to }}), venue ({{ event.venue }}), and status ({{ event.is_active }}) are displayed in list format (list-group list-group-flush).

Navigation links (Edit, Delete) allow users to manage events, directing to edit ({% url 'edit_events' event.id %}) and delete ({% url 'delete_events' event.id %}) endpoints respectively.

Create_event.html: [ix]



```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div>
        <h1>Create New Event</h1>
    </div>
    <div>
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{ form|crispy }}
            <button type="submit" class="btn btn-primary">Create</button>
        </form>
    </div>
</div>
{% endblock content %}
```

The `create_event.html` template is designed to facilitate the creation of new events through a form powered by Django's `EventForm`. This form is configured to handle HTTP POST requests (`method="post"`) and supports file uploads (`enctype="multipart/form-data"`), crucial for including a banner image.

The form fields are rendered using `{{ form|crispy }}` for enhanced styling with Bootstrap. They include title, category, start/end dates, times, description, venue, and checkboxes for expiration (`is_expired`), activation (`is_active`), and volunteer needs (`is_volunteer_required`). HTML5 input types (`type="date"`, `type="time"`) improve date and time selection. Users submit the form by clicking the "Create" button, redirecting to the `all-events` page. The `create_event.html` template leverages Django and Bootstrap for a user-friendly event creation process.

All_event.html:

```
{% extends 'base.html' %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div class="text-end">
        <a href="{% url 'create_events' %}" class="btn btn-sm btn-secondary">Create New Event</a>
    </div>
<div class="row">
    {% for event in all_events %}
    <div class="col-lg-6 col-sm-12 col-md-12">
        <div class="card" style="width: 18rem;">
            <img src="{{event.banner.url}}" class="card-img-top" alt="...">
            <div class="card-body">
              <h5 class="card-title">{{event.title}}</h5>
              <p class="card-text">{{event.description|truncatechars:200}}</p>
            </div>
            <ul class="list-group list-group-flush">
              <li class="list-group-item"><span class="badge text-bg-secondary">{{event_detail.category}}</span></li>
              <li class="list-group-item">Created On:<br>{{event.created_at}}</li>
              <li class="list-group-item">Start Date Time:<br>{{event.date_from}} - Start Time: {{event.time_from}}</li>
              <li class="list-group-item">End Date Time:<br> {{event.date_to}} - End Time: {{event.time_to}}</li>
              <li class="list-group-item">Venue:<br> {{event.venue}}</li>
              <li class="list-group-item">Status: <br>{{event.is_active}}</li>
            </ul>
            <div class="card-body">
              <a href="{% url 'edit_events' event.id %}" class="card-link">Edit</a>
              <a href="{% url 'delete_events' event.id %}" class="card-link">Delete</a>

            </div>
          </div>
    </div>
    {% empty %}
    <h4>There is no event now</h4>
    {% endfor %}
</div>
</div>
{% endblock content %}
```

The all_events.html template displays a list of nonprofit organization events using Django and Bootstrap for structure and styling. It iterates over the events with {% for event in all_events %}, presenting each within a Bootstrap card that includes the event's title, description, dates, venue, and status. In each card's body ( ), users can interact with the event by editing or deleting it via links to {% url 'edit_events' event.id %} and {% url 'delete_events' event.id %}. If no events are available, the template displays a message: <h4>There are no events available</h4>.

Edit_event.html:[x]

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div>
        <h1>Update Event</h1>
    </div>
    <div>
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{ form|crispy }}
            <button type="submit" class="btn btn-primary">Update</button>
        </form>
    </div>
</div>
{% endblock content %}
```

The edit_event.html template allows users to update details of an existing event on a nonprofit's website. When accessed, it pre-loads the event's current information into the form fields. The form uses the POST method (method="post") and supports file uploads (enctype="multipart/form-data") for managing event

details and media attachments. The form fields ({{ form|crispy }}) display data such as the title, category, date/time, description, venue, and status. Users can submit their changes by clicking the "Update" button (<button type="submit" class="btn btn-primary">Update</button>).

Event_detail.html:

```
{% extends 'base.html' %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
<h1>{{event_detail.title}}</h1>
<div class="row">
<div class="col-sm-12 col-md-12 col-lg-6">
    <img src="{{event_detail.banner.url}}" class="img-fluid" alt="" />
    <p>{{event_detail.description}}</p>
</div>
<div class="col-sm-12 col-md-12 col-lg-6">

    <div class="card">
        <div class="card-header">
         Event Details
        </div>
        <ul class="list-group list-group-flush">
          <li class="list-group-item">{{event_detail.category}}</li>
          <li class="list-group-item">Created On:{{event_detail.created_at}}</li>
          <li class="list-group-item">Start Date Time:{{event_detail.date_from}} - Start Time: {{event_detail.time_from}}</li>
          <li class="list-group-item">End Date Time: {{event_detail.date_to}} - End Time: {{event_detail.time_to}}</li>
          <li class="list-group-item">Venue: {{event_detail.venue}}</li>
          <li class="list-group-item">Status: {{event_detail.is_active}}</li>


        </ul>
      </div>
</div>
</div>
<div class="mt-2">
    <h4>Terms and Conditions</h4>
    <p>{{event_detail.terms}}</p>
</div>
<div class="mt-2">
    <h4>Volunteer</h4>
    {% for volunteer in event_detail.volunteers.all %}
    <span class="bg-warning px-2">
        <small>{{volunteer.first_name}}</small>
    </span><br>
{% endfor %}
</div>
<div class="mt-2">
    <h4>Sponsors</h4>
    {% for sponser in event_detail.sponsers.all %}
        <span class="bg-warning px-2">
            <small>{{sponser.sponser}}</small>
        </span><br>
    {% endfor %}
</div>
</div>
{% endblock content %}
```

The `event_detail.html` template shows detailed information about an event, including the title, banner image, description, category, creation date, start and end date/time, venue, and status. It also includes sections for terms and conditions, volunteers, and sponsors, displaying related objects such as volunteers and sponsors associated with the event.

Adopt Now Report Incident

EcoGuardians

Hello, Linda

Events | Donation | Blogs | News and Update | Recycle Now

Title of a longer featured blog post

Multiple lines of text that form the lede, informing new readers quickly and efficiently about what's most interesting in this post's contents.

Continue reading...

Current Events

Animal Rescue Event 2

Animal Rescue

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type ...

From : June 20, 2024 To : June 21, 2024

Animal Rescue Event 3

Animal Rescue

Description1....

From : June 20, 2024 To : June 21, 2024

Check Event

Adopt Now Report Incident

EcoGuardians

Hello, Linda

Events | Donation | Blogs | News and Update | Recycle Now

Create New Event

Animal Rescue Event 2

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type ...

Created On:
June 16, 2024, 10 a.m.

Start Date Time:
June 20, 2024 - Start Time: 12:30 p.m.

End Date Time:
June 21, 2024 - End Time: 12:30 p.m.

Venue:
Singapore

Status:
True

Edit Delete

Animal Rescue Event 3

Description1....

Created On:
June 16, 2024, 10:19 a.m.

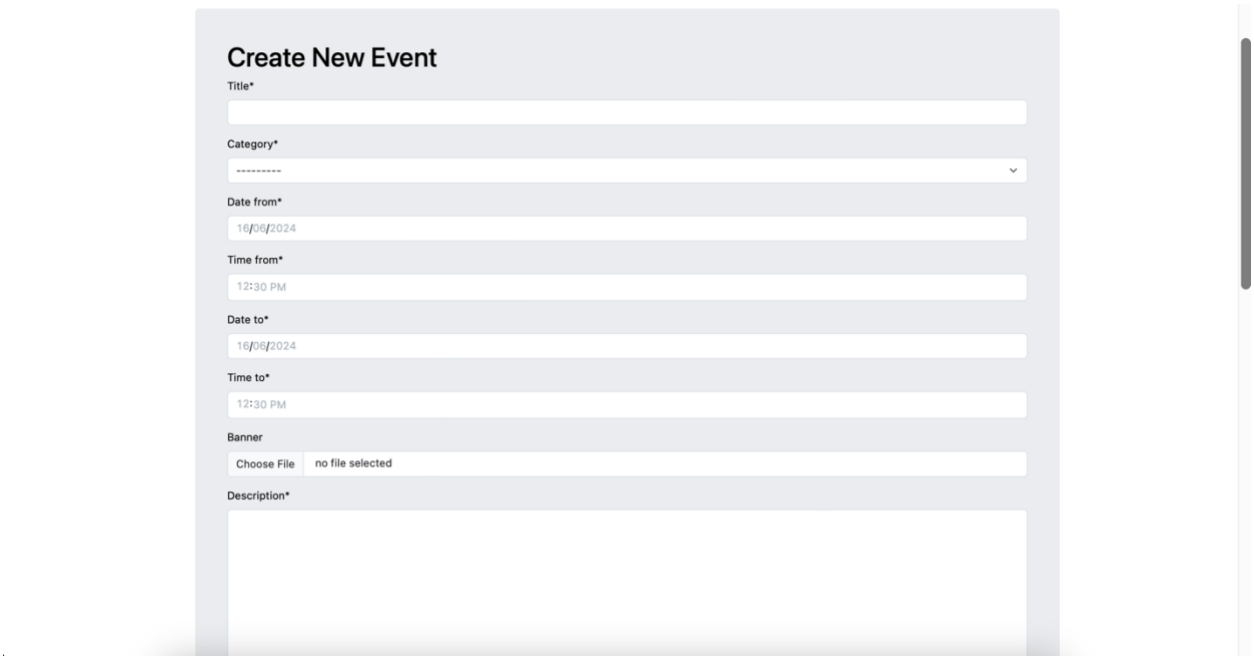
Start Date Time:
June 20, 2024 - Start Time: 1:30 p.m.

End Date Time:
June 21, 2024 - End Time: 1:30 p.m.

Venue:
Singapore

Status:
True

Edit Delete



For Volunteer App:

Forms.py:

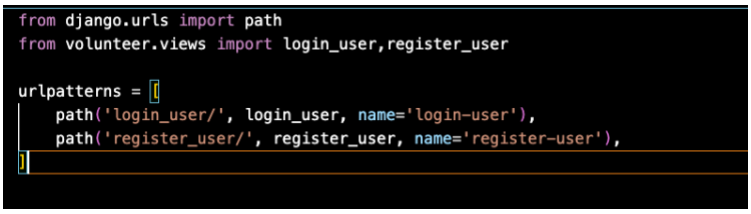
```python
from django import forms
from django.contrib.auth.models import User
from organizers.models import Volunteer


class UserForm(forms.ModelForm):
    password = forms.CharField(widget = forms.PasswordInput)

    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'email', 'username', 'password']

class VolunteerForm(forms.ModelForm):
    class Meta:
        model = Volunteer
        fields = ['gender',
                  'dob_day',
                  'dob_month',
                  'dob_year',
                  'phone',
                  'full_address',
                  'profile_pic'
                  ]
        widgets = {
            'dob_month': forms.NumberInput(attrs={'min':1,'max':12}),
            'dob_day': forms.NumberInput(attrs={'min':1,'max':31}),
            'dob_year': forms.NumberInput(attrs={'min':1900,'max':2024}),
            'phone': forms.TextInput(attrs={'maxlength':9}),
        }
class UserLoginForm(forms.Form):
    username=forms.CharField(widget=forms.TextInput(attrs={'class':'form-control', 'placeholder':'Username'}))
    password=forms.CharField(widget=forms.PasswordInput(attrs={'class':'form-control', 'placeholder':'Password'}))
```

In forms.py, several forms handle user and volunteer data. The UserForm is a ModelForm that collects basic user information like first name, last name, email, username, and password, with the password field using a PasswordInputwidget for security.

The VolunteerForm, also a ModelForm, targets the Volunteer model, capturing additional volunteer details such as gender, date of birth, phone number, address, and a profile picture. Widgets are used to restrict inputs, like limiting the date of birth range and setting a maximum length for phone numbers.

The UserLoginForm provides fields for username and password,
utilizing TextInput and PasswordInput widgets, and is used for user authentication.

Urls.py:

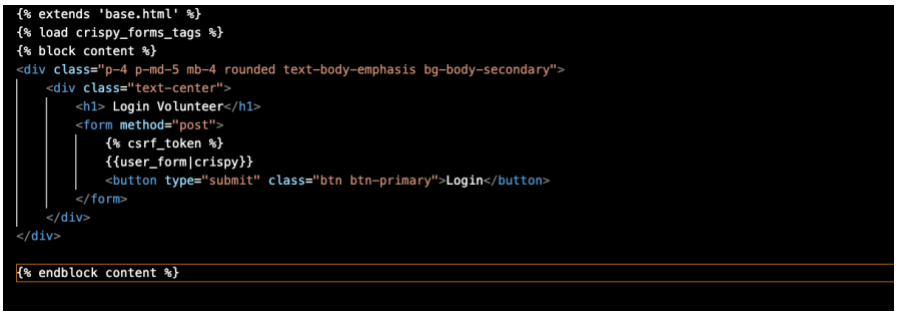


The urls.py file maps specific URL patterns to their corresponding view functions within the volunteer app. It imports the path function from django.urls and the view functions login_user and register_user from volunteer.views. Two URL patterns are defined: one for logging in (login_user/) and one for registering a new user (register_user/). Each pattern is named (login-user and register-user) to allow for easy referencing.

Views.Py:[xi]

The views.py file manages user interactions in the volunteer system. The login_user view handles authentication by processing POST requests and using Django's authenticate function. If successful, the user is logged in and redirected to the home page; otherwise, error messages are displayed for invalid credentials. The register_user view manages registration, processing
both UserForm and VolunteerForm data. Upon valid POST data, it saves the user and volunteer details, sets the password, and redirects the user to the login page after successful registration.

Login_user.html:[xii]

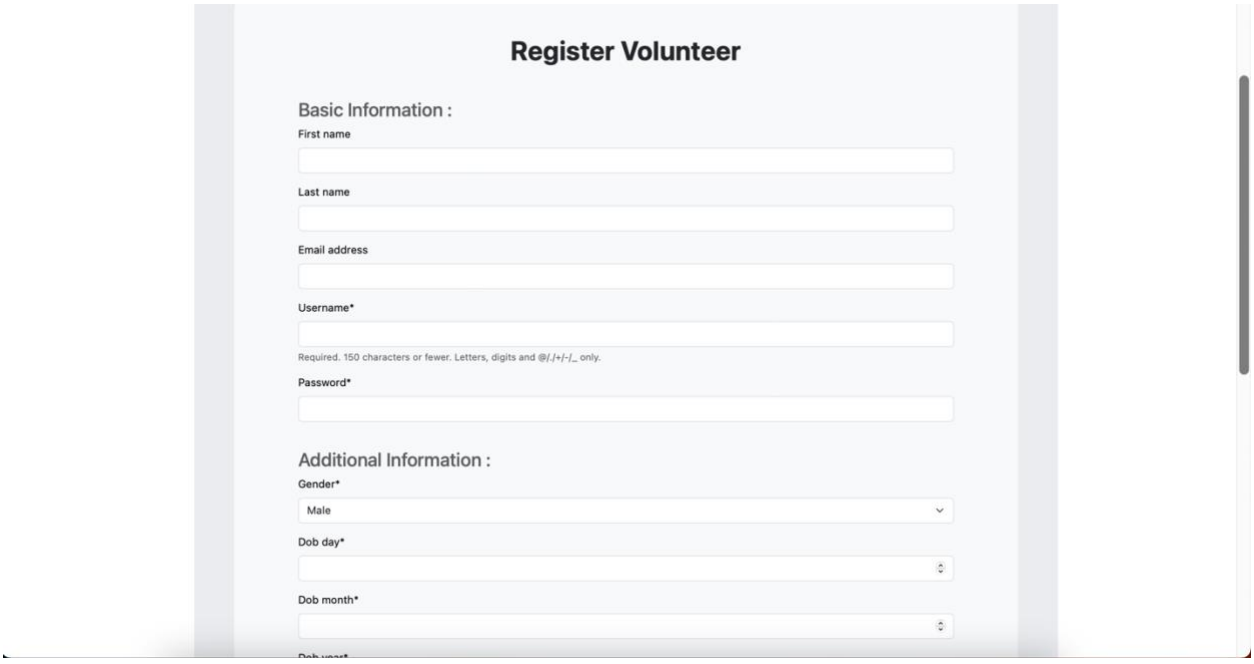


The login_user.html includes a CSRF token for security and uses the crispy template tag to render the form fields. A submit button is provided to send the login data.

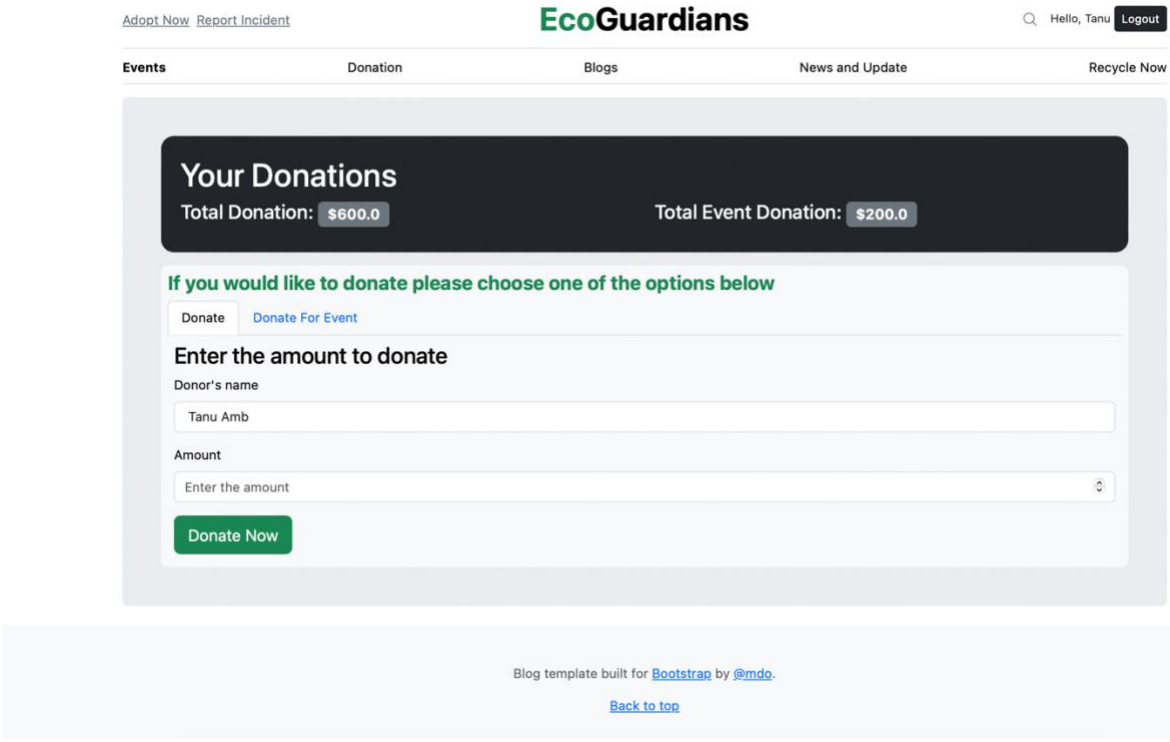
Register_user.html:

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div class="container my-5">
        <div class="p-5 bg-body-tertiary rounded-3">
            <h1 class="text-center fw-bold mb-5 text-dark">Register Volunteer</h1>
            <form method="post" enctype="multipart/form-data">
                {% csrf_token %}
                <div class="px-5">
                    <h3 class="text-muted">Basic Information :</h3>
                    {{user_form|crispy}}
                </div>
                <br>
                <div class="px-5">
                    <h3 class="text-muted">Additional Information :</h3>
                    {{volunteer_form|crispy}}
                </div>
                <button type="submit" class="btn btn-primary">Register</button>
            </form>
        </div>
    </div>
</div>
{% endblock content %}
```

The register_user.html includes a CSRF token for security and supports file uploads for a profile picture. Data is submitted to the server when the user clicks the register button.

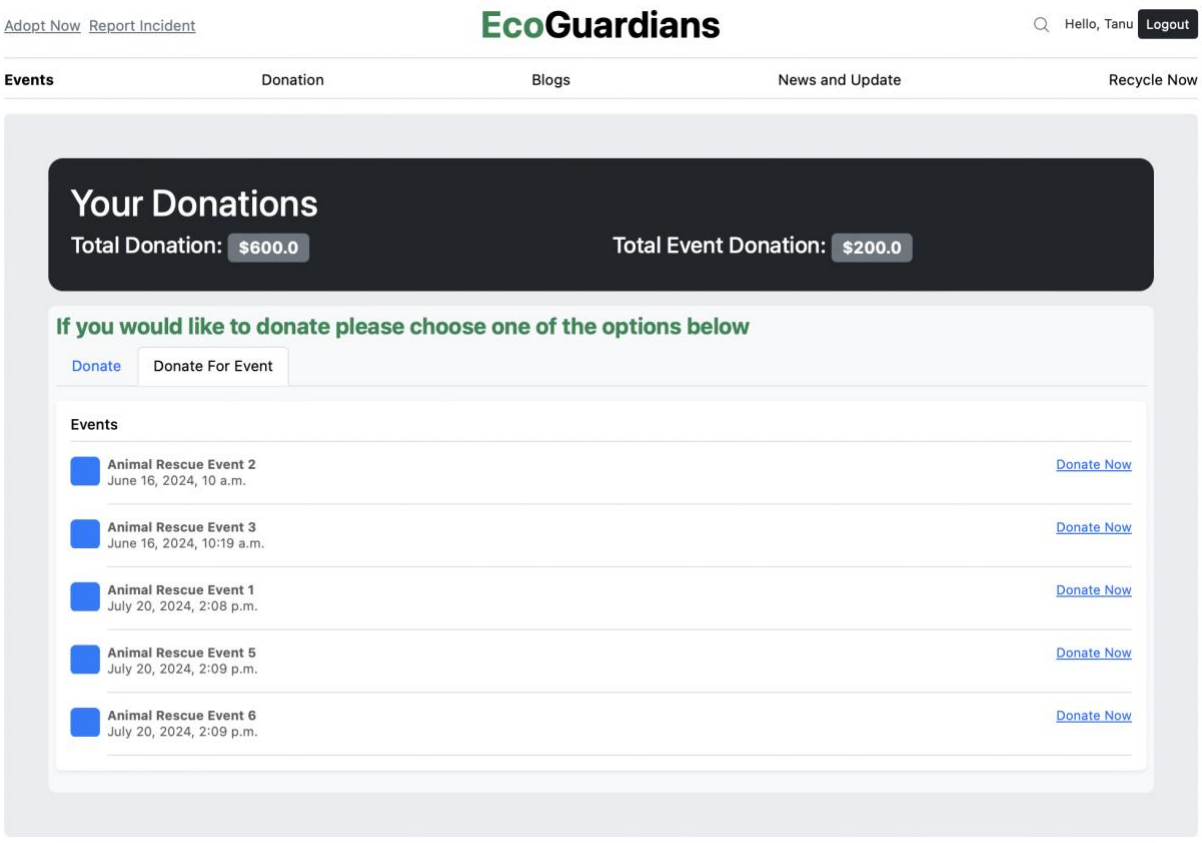


Donation Page:



Blog template built for Bootstrap by @mdo.

Back to top



Donate.html

For this html page, the code shows that it contains a form where users can enter their name and donation amount to make a general donation. The form submission is handled by the donation view.

Donate_to_event.html

For this html page, event has a link directing to its specific donation page, managed by the donation_to_event view. This layout allows users to easily choose between general donations and event-specific contributions.

Views.py

```python
def donation(request):
    if request.method=="POST":
        donor_name = request.POST["inputDonorName"].strip()
        amount = request.POST["inputAmount"].strip()

        # event_donor_name = request.POST["inputDonorNameEvent"]
        # event_amount = request.POST["inputAmountEvent"]

        if donor_name and amount:
            Donation.objects.create(donated_name = donor_name, amount = amount, if_donated_for_event=False)
            print("General donation done successfully")
            return redirect('donation')
    else:
        full_name = ''
        total_general_donation = 0.00
        total_event_donation = 0.00
        if request.user.is_authenticated:
            full_name = f"{request.user.first_name} {request.user.last_name}"
        get_all_donors = Donation.objects.filter(donated_name = full_name, if_donated_for_event= False)
        for donor in get_all_donors:
            total_general_donation += int(donor.amount)
        get_all_event_donors = Donation.objects.filter(donated_name = full_name, if_donated_for_event= True)
        for donor in get_all_event_donors:
            print("donor", donor)
            total_event_donation += int(donor.amount)
            print("total_event_donation", total_event_donation)
            #get list of events
        events = Event.objects.all()
        return render(request, 'donation/donate.html', {'get_all_donors': get_all_donors, 'total_general_donation': total_general_donation, 'events' : events, 'total_event_donation':total_event_don

def donation_to_event(request, event_slug):
    context={}
    event = Event.objects.filter(slug = event_slug)
    if event.exists():
        event_obj = Event.objects.get(slug = event_slug)
        context["event"] = event_obj

    if request.method=="POST":
        event_donated_name = request.POST["inputDonorNameEvent"].strip()
        event_amount = request.POST["inputAmountEvent"]
        if event_donated_name and event_amount:
            Donation.objects.create(donated_name = event_donated_name, amount = event_amount, if_donated_for_event=True, event=event_obj)
            print("Event donation done successfully")
            return redirect('donation')
    else:
        return render(request, 'donation/donate_to_event.html', context)
```

```python
#donation urls
path('donation/', donation, name='donation'),
path('donation-to-event/<str:event_slug>/', donation_to_event, name='donation_to_event'),
```

The donation view manages general donations and displays relevant details. For POST requests, it captures the donor's name and donation amount from the form. If valid, a new Donation object is created, and the user is redirected, ensuring the donation is recorded. For GET requests, it calculates the total general and event-specific donations for the authenticated user by filtering based on if_donated_for_event. It also fetches available events, passing this data to the donate.html template to show the user's donation history and upcoming events.

The donation_to_event view retrieves the event by its slug. Upon form submission, it captures the donor's name and amount, creating a new Donation linked to the event.

Add to Calendar Page: [xiii]

Animal Rescue Event 1

Animal Rescue Event

Created On:
July 20, 2024, 2:08 p.m.

Start Date Time:
July 21, 2024 - Start Time: 1 p.m.

End Date Time:
July 23, 2024 - End Time: 1 p.m.

Venue:
Singapore

Status:
True

Ongoing

Add to calandar

Animal Rescue Event 5

Animal Rescue Event

Created On:
July 20, 2024, 2:09 p.m.

Start Date Time:
July 21, 2024 - Start Time: 1 p.m.

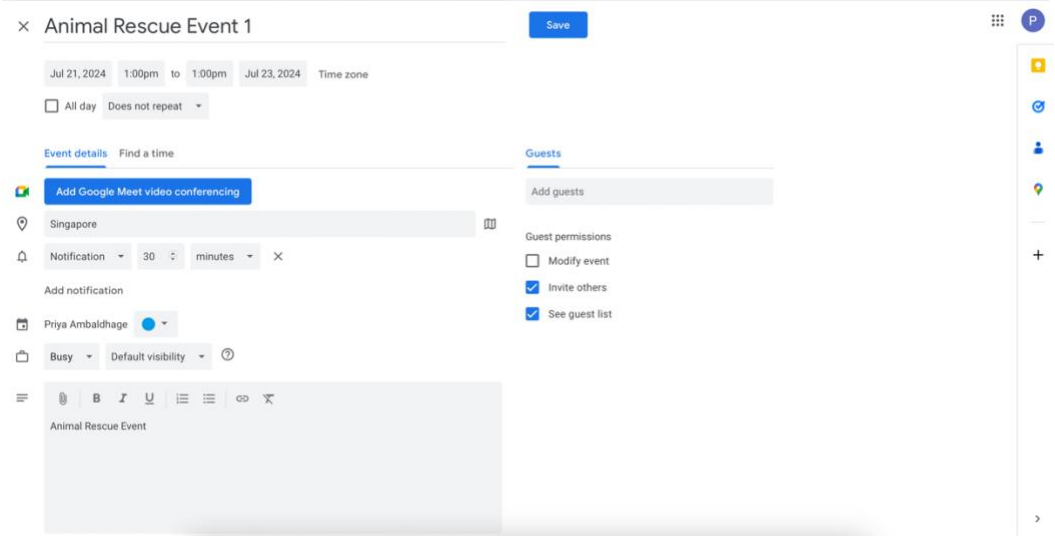
End Date Time:
July 23, 2024 - End Time: 1 p.m.

Venue:
Singapore

Status:
True

Ongoing

Add to calandar



```python
def event_details(request, event_slug):
    context = {}
    event_detail = Event.objects.get(slug=event_slug)

    combine_start_date_time = datetime.strptime(f'{event_detail.date_from} {event_detail.time_from}', '%Y-%m-%d %H:%M:%S')
    combine_end_date_time = datetime.strptime(f'{event_detail.date_to} {event_detail.time_to}', '%Y-%m-%d %H:%M:%S')
    print(combine_start_date_time, '>>>>>>>>>combined start date time') #2024-06-22 23:45:00
    print(combine_end_date_time, '>>>>>>>>>combined end date time') #2024-06-22 23:45:00

    # format the datetime obj
    start_formatted_dattime = combine_start_date_time.strftime('%Y%m%dT%H%M%S')
    end_formatted_dattime = combine_end_date_time.strftime('%Y%m%dT%H%M%S')

    params = {
        'action':'TEMPLATE',
        'text':event_detail.title,
        'dates':f'{start_formatted_dattime}/{end_formatted_dattime}',
        'details': event_detail.description,
        'location': event_detail.venue,
        'trp':'false',
        'sprop':''
    }

    google_calender_url = f'https://calendar.google.com/calendar/render?{urlencode(params)}'
    context["event_detail"] = event_detail
    context['google_calender_url'] = google_calender_url
    return render(request, 'organizers/event_detail.html', context)
```

```python
def all_events(request):
    context = {}
    all_events = Event.objects.all()
    event_links = [] # [(event1, calanderurl1), (event2, calenderurl2), (event3, calenderurl3)]

    for event in all_events:
        google_calander_url = create_google_calendar_url(event)
        #combine the date and time
        event_date_time = datetime.combine(event.date_from, event.time_from)
        #check if the event date and time is in the past, future
        is_future_event = event_date_time > datetime.now()
        print(is_future_event, ">>>>>printing if future event")
        event_links.append((event, google_calander_url, is_future_event))

    print(event_links,">>>>prepared event link")
    # o/p: [(<Event: Test Event one>, 'https://calendar.google.com/calendar/render?action=TEMPLATE&text=Test+Event+one&dates=20240622T234500%2F20240524T235200&details=Test+Event&location=Begumpet+Hyderabad&trp=false&sprop=')]
    context['all_events'] = event_links

    return render(request, 'organizers/all_events.html', context)
```

```python
# creating a function to prepare the google calander url
def create_google_calendar_url(event):
    # formate the datetime in this way '%Y%m%dT%H%M%S'
    combine_start_date_time = datetime.strptime(f'{event.date_from} {event.time_from}', '%Y-%m-%d %H:%M:%S')
    combine_end_date_time = datetime.strptime(f'{event.date_to} {event.time_to}', '%Y-%m-%d %H:%M:%S')
    print(combine_start_date_time, '>>>>>>>>>combined start date time') #2024-06-22 23:45:00
    print(combine_end_date_time, '>>>>>>>>>combined end date time') #2024-06-22 23:45:00

    # format the datetime obj
    start_formatted_dattime = combine_start_date_time.strftime('%Y%m%dT%H%M%S')
    end_formatted_dattime = combine_end_date_time.strftime('%Y%m%dT%H%M%S')

    params = {
        'action':'TEMPLATE',
        'text':event.title,
        'dates':f'{start_formatted_dattime}/{end_formatted_dattime}',
        'details': event.description,
        'location': event.venue,
        'trp':'false',
        'sprop':''
    }

    # https://calendar.google.com/calendar/render?action=TEMPLATE&text=TEST+EVENT=1
    # o/p: https://calendar.google.com/calendar/render?action=TEMPLATE&text=Test+Event+one&dates=20240622T234500%2F20240524T235200&details=Test+Event&location=Begumpet+Hyderabad&trp=false&sprop=
    return f'https://calendar.google.com/calendar/render?{urlencode(params)}'
```

The event_details view displays detailed information about a specific event and provides a Google Calendar link for easy addition. It retrieves the event using its slug, formats the start and end dates into datetime objects, and constructs a Google Calendar URL by encoding the event's title, dates, description, and location, enabling users to add the event to their calendar with one click.

The all_events view lists all events and generates Google Calendar links for future events.
The create_google_calendar_url helper function formats the event's details into the required format and returns the URL for effortless calendar addition.

Adopt Now Page:[xiv]

Model.js

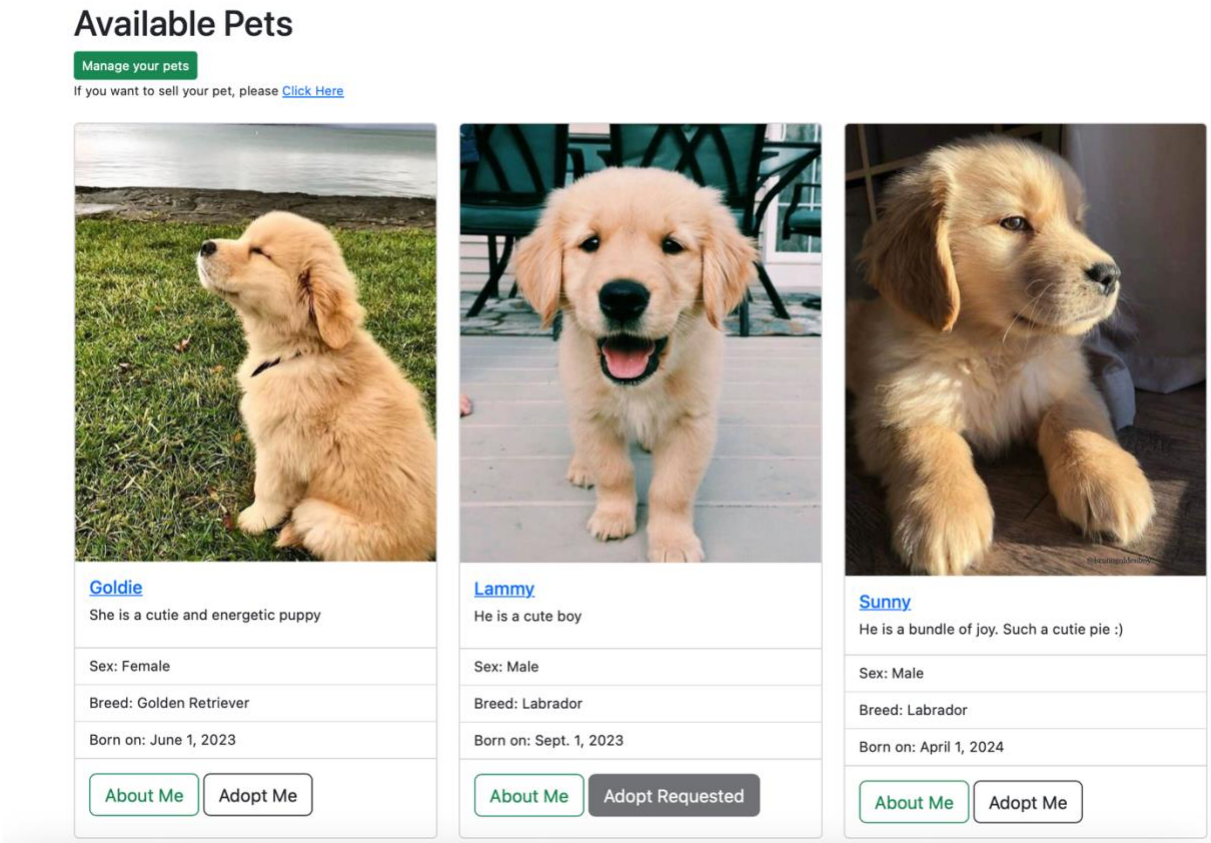
```python
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone

# Create your models here.

class PetBreed(models.Model):
    name = models.CharField(max_length=100)
    slug = models.SlugField(unique = True, blank=True, null=True)
    description = models.TextField()
    size = models.IntegerField(default=0)
    lifespan = models.IntegerField(default=0)
    trainability = models.IntegerField(default=0)
    bark = models.IntegerField(default=0)
    energy = models.IntegerField(default=0)
    image = models.ImageField(upload_to='breed_images/{name}/', blank=True, null=True)

    def __str__(self):
        return self.name


class Pet(models.Model):
    SEX_CHOICE = (
        ("male", 'MALE'),
        ("female", 'FEMALE')
    )
    name = models.CharField(max_length=100)
    slug = models.SlugField(unique = True, blank=True, null=True)
    owner_name = models.ForeignKey(User, on_delete=models.CASCADE)
    breed = models.ForeignKey(PetBreed, on_delete=models.CASCADE)
    sex = models.CharField(max_length=50)
    price = models.FloatField(default=0.00)
    location = models.CharField(max_length=500)
    contact = models.CharField(max_length=9)
    dob = models.DateField()
    description = models.TextField()
    image = models.ImageField(upload_to='pet_images/{name}/', blank=True, null=True)

    def __str__(self):
        return self.name


class PetContact(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    pet = models.ForeignKey(Pet, on_delete=models.CASCADE)
    email = models.EmailField()
    contact = models.CharField(max_length=9)
    message = models.TextField()

    def __str__(self):
        return f"{self.first_name} {self.last_name}"


class AdoptionStatus(models.Model):
    adoptee = models.ForeignKey(PetContact, on_delete=models.CASCADE)
    pet = models.ForeignKey(Pet, on_delete=models.CASCADE)
    pet_adopt_request = models.BooleanField(default=False)
    adoption_completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Urls.py

```python
from django.urls import path
from adopt.views import list_pets, list_breeds, create_pet, detail_pet, adopt_pet, adoption_success, manage_pets, edit_pet, delete_pet, adopt_requests, request_accepted, request_

urlpatterns = [
    #path('',  home, name='home')
    path('list-pets', list_pets, name='list-pets'),
    path('create-pet', create_pet, name='create-pet'),
    path('edit-pet/<str:pet_slug>/', edit_pet, name='edit-pet'),
    path('delete-pet/<str:pet_slug>/', delete_pet, name='delete-pet'),
    path('list-breeds', list_breeds, name='list-breeds'),
    path('<str:pet_slug>/', detail_pet, name='detail-pet'),
    path('<slug:slug>/adopt/', adopt_pet, name='adopt-pet'),
    path('adoption-success/', adoption_success, name='adoption_success'),
    path('manage/<int:user_id>/pets', manage_pets, name='manage-pets'),
    path('manage/<int:user_id>/adopt-requests/', adopt_requests, name='adopt-requests'),
    path('<int:request_id>/<int:user_id>/request-accepted/', request_accepted, name='request-accepted'),
    path('<int:request_id>/<int:user_id>/request-cancelled/', request_cancelled, name='request-cancelled')
]
```

list_pets.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="container mt-5">
    <h1>Available Pets</h1>
    <a href="{% url 'manage-pets' request.user.id %}" class="btn btn-sm btn-success">Manage your pets</a><br>
    <small>If you want to sell your pet, please <a href="{% url 'create-pet' %}">Click Here</a></small>
    <div class="row mt-4">
        {% for pet in pets %}
        <div class="col-sm-12 col-md-6 col-lg-4 mb-4 d-flex align-items-stretch">
            <div class="card">
                <div class="image-container">
                    <img src="{{ pet.image.url }}" class="card-img-top" alt="{{ pet.name }}_img">
                </div>
                <div class="card-body">
                    <h5 class="card-title"><a href="{% url 'detail-pet' pet.slug %}">{{ pet.name }}</a></h5>
                    <p class="card-text">{{ pet.description|truncatechars:80 }}</p>
                </div>
                <ul class="list-group list-group-flush">
                    <li class="list-group-item">Sex: {{ pet.sex|title }}</li>
                    <li class="list-group-item">Breed: {{ pet.breed.name }}</li>
                    <li class="list-group-item">Born on: {{ pet.dob }}</li>
                </ul>
                <div class="card-body">
                    <a href="{% url 'detail-pet' pet.slug %}" class="btn btn-lg rounded-3 btn-outline-success">About Me</a>
                    {% if pet.id in adopted_pet_ids %}
                    <button disabled class="btn btn-lg rounded-3 btn-dark">Adopt Requested</a>
                    {% else %}
                    <a href="{% url 'adopt-pet' pet.slug %}" class="btn btn-lg rounded-3 btn-outline-dark">Adopt Me</a>
                    {% endif %}
                </div>
            </div>
        </div>
        {% empty %}
        <small>There are no pets available to adopt!</small>
        {% endfor %}
    </div>
</div>
{% endblock content %}
```



---

detail_pet.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <h1>Detail Pet: {{ pet.name }}</h1>
</div>

<div class="container mt-4">
    <div class="row">
        <div class="col-md-6">
            <div class="image-container mb-4">
                <img src="{{ pet.image.url }}" class="img-fluid rounded" alt="{{ pet.name }}">
            </div>
        </div>
        <div class="col-md-6">
            <h2>{{ pet.name }}</h2>
            <h4>Breed: {{ pet.breed.name }}</h4>
            <p>{{ pet.description }}</p>
            <ul class="list-group list-group-flush">
                <li class="list-group-item"><strong><i class="fas fa-user"></i> Owner:</strong> {{ pet.owner_name }}</li>
                <li class="list-group-item"><strong><i class="fas fa-venus-mars"></i> Sex:</strong> {{ pet.sex|title }}</li>
                <li class="list-group-item"><strong><i class="fas fa-calendar-alt"></i> Date of Birth:</strong> {{ pet.dob }}</li>
                <li class="list-group-item"><strong><i class="fas fa-map-marker-alt"></i> Location:</strong> {{ pet.location }}</li>
                <li class="list-group-item"><strong><i class="fas fa-dollar-sign"></i> Price:</strong> ${{ pet.price }}</li>
                <li class="list-group-item"><strong><i class="fas fa-phone"></i> Contact:</strong> {{ pet.contact }}</li>
            </ul>
            <div class="mt-4">
                {% if pet.id in adopted_pet_ids %}
                <small class="text-success fw-bold">Adoption request is already made for this pet!</small>
                {% else %}
                <a href="{% url 'adopt-pet' pet.slug %}" class="btn btn-lg rounded-3 btn-outline-success">Adopt Me</a>
                {% endif %}
            </div>
        </div>
    </div>
    <div class="row mt-5">
        <div class="col-12">
            <h3>Breed Details</h3>
            <p>{{ pet.breed.description }}</p>
            <ul class="list-group list-group-flush">
                <li class="list-group-item"><strong>Size:</strong> {{ pet.breed.size }}</li>
                <li class="list-group-item"><strong>Lifespan:</strong> {{ pet.breed.lifespan }} years</li>
                <li class="list-group-item"><strong>Trainability:</strong> {{ pet.breed.trainability }}</li>
                <li class="list-group-item"><strong>Bark Level:</strong> {{ pet.breed.bark }}</li>
                <li class="list-group-item"><strong>Energy Level:</strong> {{ pet.breed.energy }}</li>
            </ul>
        </div>
    </div>
</div>
{% endblock content %}
```

Detail Pet: Lammy

Lammy

Breed: Labrador

He is a cute boy

- **Owner:** Dhruv
- **Sex:** Male
- **Date of Birth:** Sept. 1, 2023
- **Location:** Singapore
- **Price:** $400.0
- **Contact:** 91234567

Adoption request is already made for this pet!

Breed Details

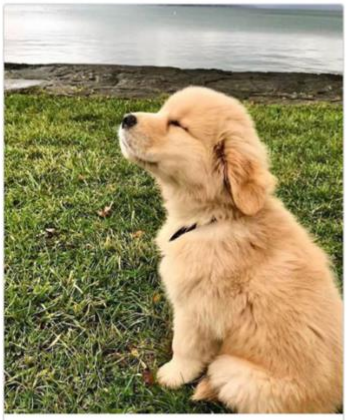
Labrador Retrievers (commonly known as Labradors) are sweet-faced, loveable and adorable! One of the most popular dog breeds in India, Labradors are outgoing, friendly, and have enough love in them for the whole family! The Labrador, a great socialiser, rarely has a bone to pick with neighbouring dogs.

- **Size:** 60
- **Lifespan:** 13 years
- **Trainability:** 80
- **Bark Level:** 60
- **Energy Level:** 70

Adopt Now Report Incident **EcoGuardians** Hello, Tanu Logout

Events Donation Community Forum News and Update Recycle Now

Adopt Goldie



Goldie

Breed: Golden Retriever

Date of Birth: June 1, 2023

She is a cutie and energetic puppy

Adoption Request

First name*

Last name*

Email*

Contact*

Message*

Why do you want to adopt this pet?

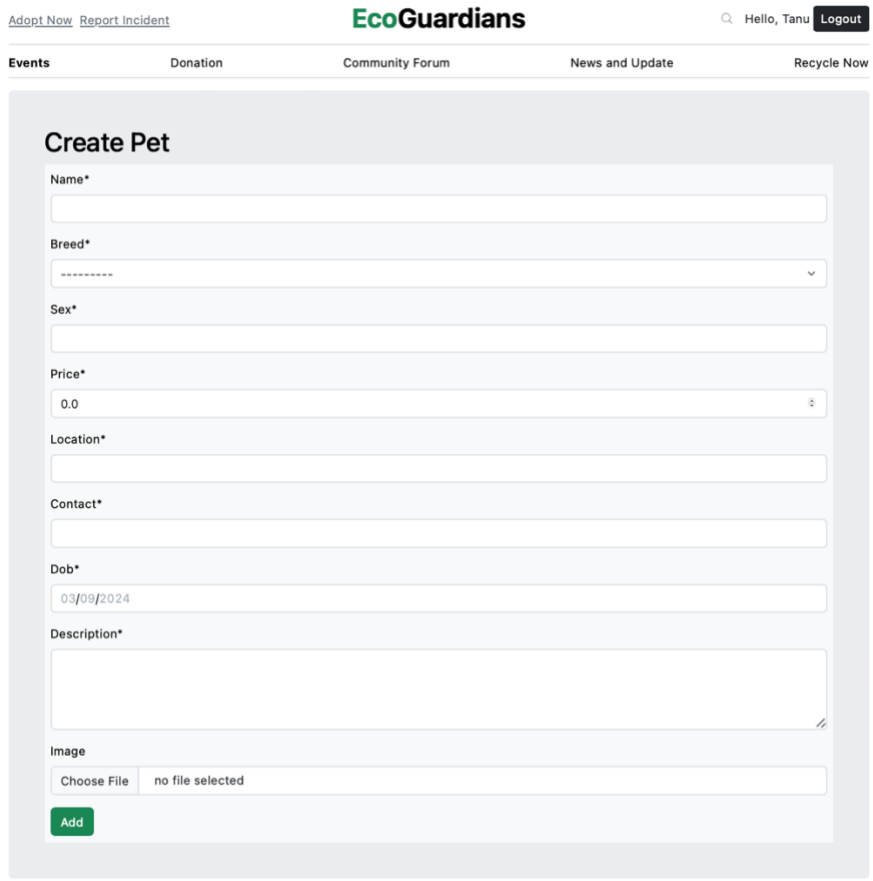
Submit Request

Blog template built for Bootstrap by @mdo.

Back to top

Adopt_me.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="container mt-5">
    <h1>Adopt {{ pet.name }}</h1>
    <div class="row mt-4">
        <!-- Pet Details -->
        <div class="col-md-6">
            <div class="card mb-4">
                <img src="{{ pet.image.url }}" class="card-img-top" alt="{{ pet.name }}">
                <div class="card-body">
                    <h4 class="card-title">{{ pet.name }}</h4>
                    <p class="card-text">Breed: {{ pet.breed.name }}</p>
                    <p class="card-text">Date of Birth: {{ pet.dob }}</p>
                    <p class="card-text">{{ pet.description }}</p>
                </div>
            </div>
        </div>
        <!-- Adoption Request Form -->
        <div class="col-md-6">
            <h2>Adoption Request</h2>
            <form method="post">
                {% csrf_token %}

                <!-- Display form errors -->
                {% if form.errors %}
                <div class="alert alert-danger mt-3">
                    <ul>
                        {% for field in form %}
                            {% for error in field.errors %}
                                <li>{{ error }}</li>
                            {% endfor %}
                        {% endfor %}
                        {% for error in form.non_field_errors %}
                            <li>{{ error }}</li>
                        {% endfor %}
                    </ul>
                </div>
                {% endif %}

                {{ form|crispy }}
                <button type="submit" class="btn btn-lg rounded-3 btn-outline-success mt-3">Submit Request</button>
            </form>
        </div>
    </div>
</div>
{% endblock content %}
```



Create_pet.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">

    <h1> Create Pet</h1>
    <div class="p-2 bg-light">
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{form|crispy}}
            <button class="btn btn-success">Add</button>
        </form>
    </div>

{% endblock content %}
```

Forms.py

```
from django import forms
from adopt.models import Pet, PetBreed, PetContact

class PetForm(forms.ModelForm):
    class Meta:
        model = Pet
        fields = '__all__'
        exclude = ['slug', 'owner_name']
        widgets = {
            'dob': forms.DateInput(attrs={'type': 'date'}),
            'description': forms.Textarea(attrs={'rows': 4}),
            'image': forms.ClearableFileInput(attrs={'class': 'form-control'})
        }

class AdoptionForm(forms.ModelForm):
    first_name = forms.CharField(max_length=100)
    last_name = forms.CharField(max_length=100)
    email = forms.EmailField()
    contact = forms.CharField(max_length=15)
    message = forms.CharField(widget=forms.Textarea, help_text="Why do you want to adopt this pet?")

    class Meta:
        model = PetContact
        fields = ['first_name','last_name', 'email', 'contact', 'message']


class PetContactForm(forms.ModelForm):
    class Meta:
        model = PetContact
        fields = ['first_name', 'last_name', 'email', 'contact', 'message']
```

Views.py

```python
def list_pets(request):
    context= {}
    if request.user.is_authenticated:
        pets = Pet.objects.exclude(owner_name=request.user)
        context['pets'] = pets
    else:
        pets = Pet.objects.all()
        context['pets'] = pets

    adopted_pet_ids = AdoptionStatus.objects.filter(pet_adopt_request=True, adoption_completed=False).values_list('pet__id', flat=True)
    context['adopted_pet_ids'] = adopted_pet_ids
    return render(request, 'pets/list_pets.html', context)


def create_pet(request):
    if request.method == 'POST':
        form = PetForm(request.POST, request.FILES)
        if form.is_valid():
            instance=form.save(commit=False)
            instance.owner_name = request.user
            instance.save()
            return redirect('list-pets') #redirect to the list pets if new pets is added
    else:
        form = PetForm()
    return render(request, 'pets/create_pet.html', {'form': form})

def edit_pet(request, pet_slug):
    pet_obj = get_object_or_404(Pet, slug=pet_slug)
    if request.method == 'POST':
        form = PetForm(request.POST, request.FILES, instance=pet_obj)
        if form.is_valid():
            instance=form.save(commit=False)
            instance.owner_name = request.user
            instance.save()
            return redirect('manage-pets', user_id=request.user.id) #redirect to the list pets if new pets is added
    else:
        form = PetForm(instance=pet_obj)
    return render(request, 'pets/edit_pet.html', {'form': form})

def delete_pet(request, pet_slug):
    pet_obj = get_object_or_404(Pet, slug=pet_slug)
    pet_obj.delete()
    return redirect('manage-pets', user_id=request.user.id) #redirect to the list pets if new pets is added


def list_breeds(request):
    pet_breeds = PetBreed.objects.all()
    context = {
        'pet_breeds': pet_breeds
    }
    return render(request,'pet_breeds/list_breeds.html', context)
```

```python
def detail_pet(request, pet_slug):
    pet = get_object_or_404(Pet, slug=pet_slug)
    print(pet)  # Debugging: Check if the pet is correctly fetched
    context = {
        'pet': pet,
        'pet_images': pet.image  # If you need to use pet_images separately
    }
    adopted_pet_ids = AdoptionStatus.objects.filter(pet_adopt_request=True, adoption_completed=False).values_list('pet__id', flat=True)
    context['adopted_pet_ids'] = adopted_pet_ids
    return render(request, 'pets/detail_pet.html', context)


def adopt_pet(request, slug):
    pet = get_object_or_404(Pet, slug=slug)
    if AdoptionStatus.objects.filter(pet_adopt_request=True, pet=pet).exists():
        return redirect('list-pets')
    else:
        if request.method == 'POST':
            form = AdoptionForm(request.POST)
            if form.is_valid():
                instance = form.save(commit=False)
                instance.pet = pet
                instance.save()
                AdoptionStatus.objects.create(adoptee=instance, pet=pet, pet_adopt_request=True)
                send_mail(
                    "Adoption Request Update",
                    f"Thank you for showing interest to adopt {pet.name}. We will send you a confirmation email in 2-3 days.",
                    "ecoguardian@gmail.com",
                    [instance.email],
                    fail_silently=False,
                )
                # handle form submission (e.g., send an email)
                return render(request, 'pets/adoption_success.html')
            else:
                form = AdoptionForm()
            return render(request, 'pets/adopt_me.html', {'form': form, 'pet': pet})

def adoption_success(request):
    return render(request, 'pets/adoption_success.html')


def manage_pets(request, user_id):
    if request.user.is_authenticated:
        user = get_object_or_404(User, id=user_id)

        pets = Pet.objects.filter(owner_name = user)
        return render(request, 'pets/manage_pets.html', {'pets': pets})

def adopt_requests(request, user_id):
    context = {}
    if request.user.is_authenticated:
        user = get_object_or_404(User,id=user_id)
        adopt_requests = AdoptionStatus.objects.filter(pet__owner_name=user)
        context['adopt_requests'] = adopt_requests

        return render(request, 'pets/adopt_request.html', context)
```

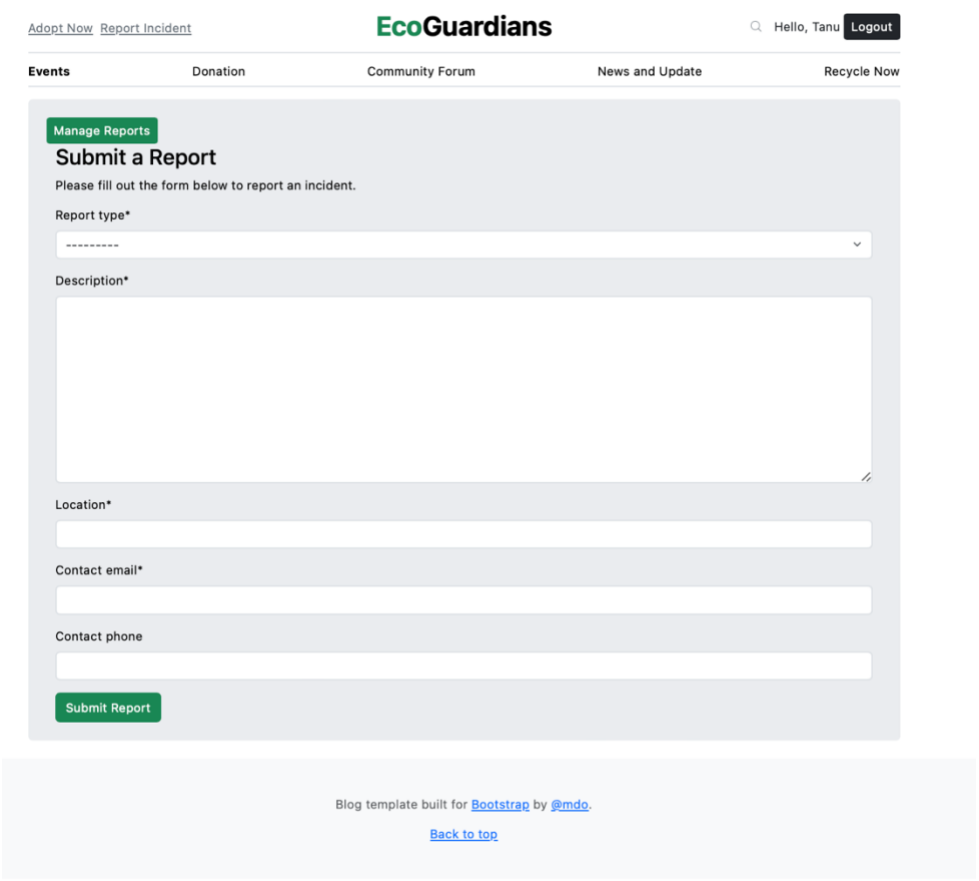
```python
def request_accepted(request, request_id, user_id):
    req = get_object_or_404(AdoptionStatus, id=request_id)
    req.adoption_completed = True
    req.save()
    send_mail(
                "Adoption Request Accepted",
                f"This is an confirmation email about the aoption request you have made for the {req.pet.name}. You can adopt {req.pet.name}. Please reply back for further communicat
                "ecoguardian@gmail.com",
                [req.adoptee.email],
                fail_silently=False,
            )
    return redirect('adopt-requests', user_id= user_id)


def request_cancelled(request, request_id, user_id):
    req = get_object_or_404(AdoptionStatus, id=request_id)
    req.adoption_completed = False
    req.save()
    send_mail(
                "Adoption Request Cancelled",
                f"This is an confirmation email about the aoption request you have made for the {req.pet.name}. You cannot adopt {req.pet.name}. Please kindly check out other availat
                "ecoguardian@gmail.com",
                [req.adoptee.email],
                fail_silently=False,
            )
    return redirect('adopt-requests', user_id= user_id)
```

This feature allows users to list their pets for adoption and manage these listings by editing or deleting them. Potential adopters can browse available pets, submit adoption requests, or make direct purchases through the platform. To ensure security, adopters must create an account before making a request or purchase. Once a request is made, adopters can track its status, including whether it has been confirmed.

Pet listers can review adoption requests, accepting or canceling them as needed, giving them full control over the process. Both parties receive email notifications about the status and actions taken, enhancing communication and transparency.

This solution provides a user-friendly experience for pet owners and adopters, combining account management, request tracking, and notifications into an interactive and engaging platform.

Report Now Page:



All Reports 2

Pet Violence

Hello someone at blk xxx is abusing a cat for the past 2-3 days

Address: Singapore
Reporter Email: priyaambaldhage@gmail.com
Reporter Contact: 91234567
Reported On: Sept. 3, 2024, 10:17 a.m.

Not Completed

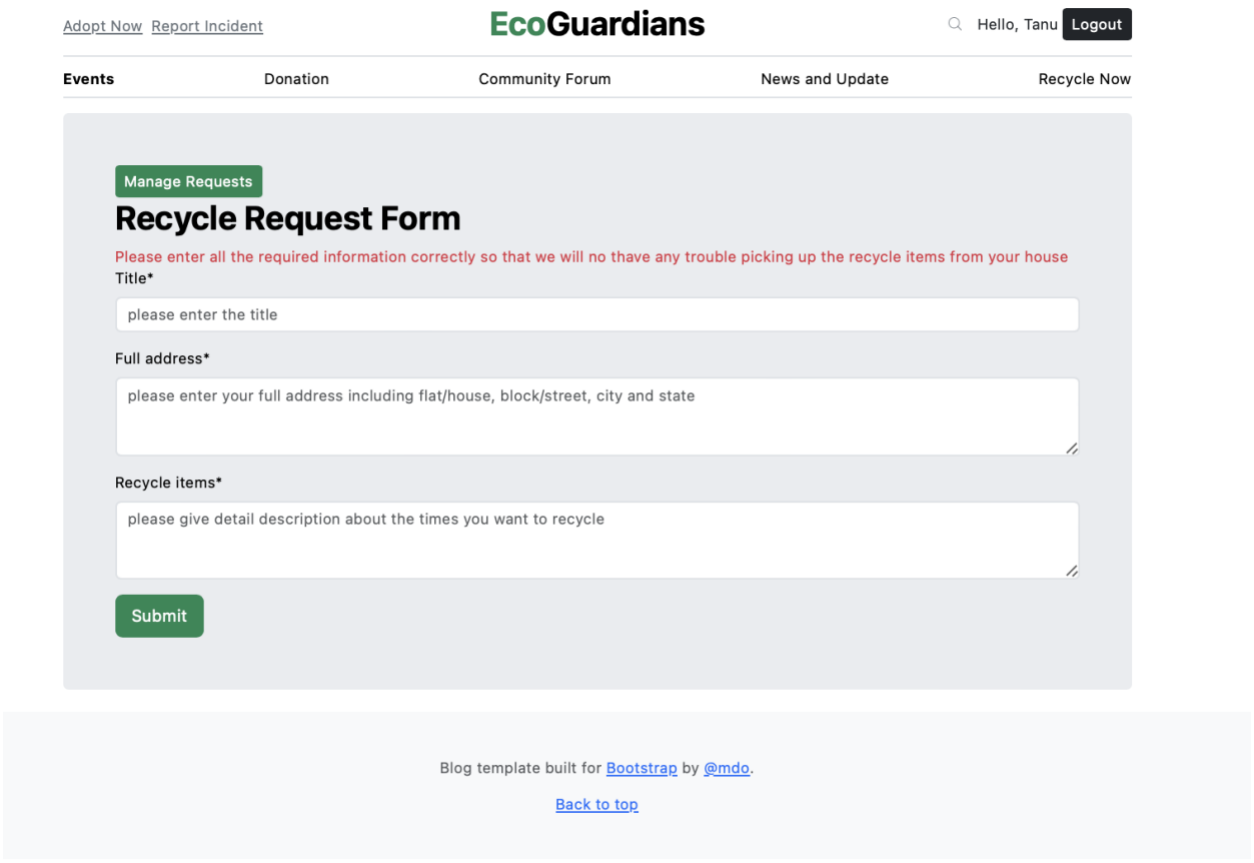
Pet Violence

...

Address: Singapore
Reporter Email: priyaambaldhage@gmail.com
Reporter Contact: 91234567
Reported On: Sept. 3, 2024, 10:18 a.m.

Completed

Recycle Now Page:



All Recycling Requests 3

When you accept or reject, user will get an comfirmation email. If accepted, user will get notified that their items will be collected in 2-3 days followed by another email for collection date

Old Sofa

Recycling Items: Old Sofa

Full Name: Tanu Amb

Full Address: Singapore

Email Address: Tanu@gmail.com

Contact:

Accept

Sept. 3, 2024, 8:19 a.m.

Recycle old sofa

Recycling Items: Old sofa

Full Name:

Full Address: Singapore

Email Address: priyaambaldhage@gmail.com

Contact: 81234567

Accept

Sept. 3, 2024, 8:18 a.m.

Recycle Sofa

Recycling Items: Old sofa

Full Name: Tanu Amb

Full Address: Singapore

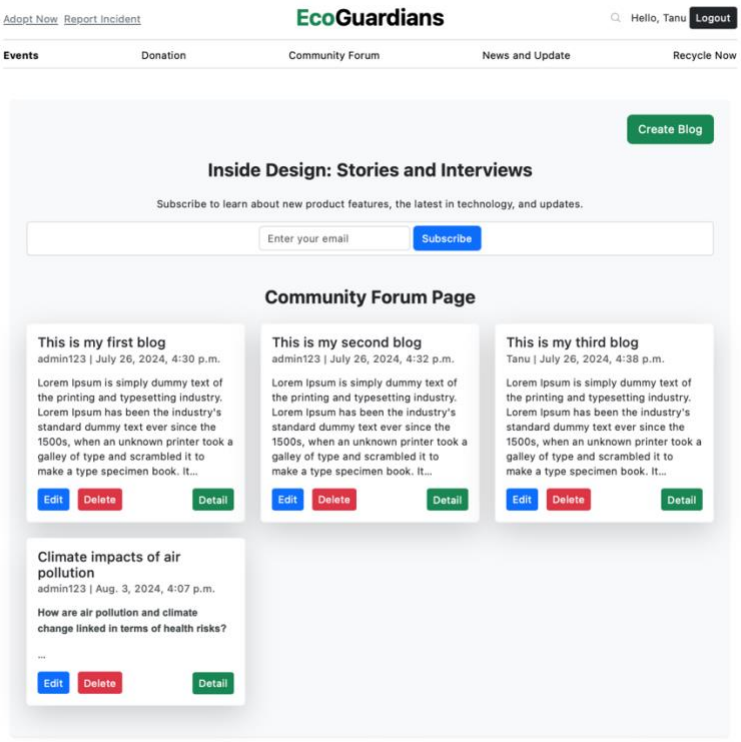
Email Address: Tanu@gmail.com

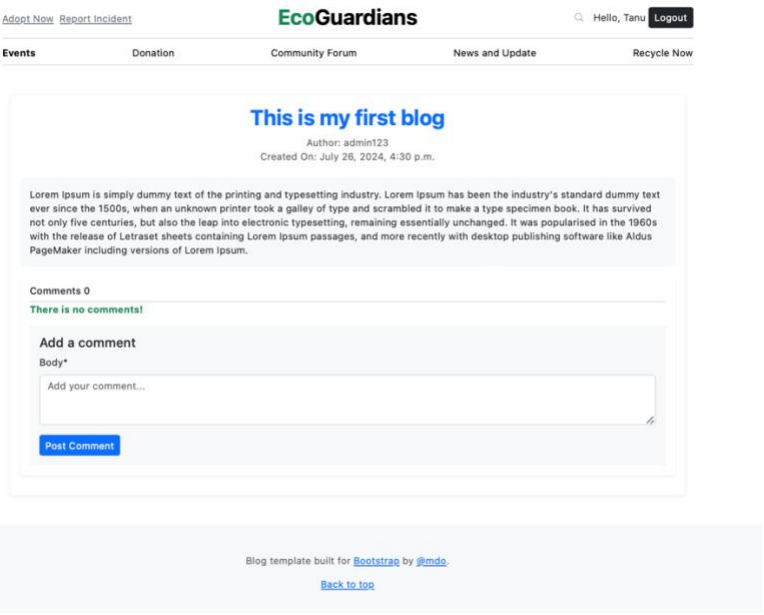
Contact:

Reject

Sept. 2, 2024, 6:08 p.m.

Community Forum Page: [xv] [xvi]





Models.py

```python
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class Blog(models.Model):
    title = models.CharField(max_length = 200, unique=True)
    slug = models.SlugField(unique = True, blank=True, null=True)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add = True)
    updated_at = models.DateTimeField(auto_now = True)

    def __str__(self):
        return self.title

class Subscriber(models.Model):
    email = models.EmailField(max_length=360)
    subscribed_on = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.email

class Comment(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE, blank=True, null=True)
    full_name = models.CharField(max_length=200, blank=True, null=True)
    email = models.EmailField(blank=True, null=True)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Comment By{self.full_name if not self.user else self.user.username} on {self.blog.title}"
```

Urls.py

```python
from django.urls import path
from blog.views import create_blog, list_blogs, detail_blog, edit_blog, delete_blog

urlpatterns = [
    path('create_blog/', create_blog, name='create-blog'),
    path('edit_blog/<str:blog_slug>/', edit_blog, name='edit-blog'),
    path('delete_blog/<str:blog_slug>/', delete_blog, name='delete-blog'),
    path('list_blogs/', list_blogs, name='list-blogs'),
    path('<str:blog_slug>/', detail_blog, name='detail-blog'),
]
```

Views.py[xvii]

```python
# Create your views here.
def create_blog(request):
    context = {}
    if request.method == 'POST':
        form = BlogForm(request.POST)
        if form.is_valid():
            blog = form.save(commit= False)
            blog.author = request.user
            blog.save()
            messages.success(request, 'Blog created successfully!')
            return redirect('list-blogs')
        else:
            messages.error(request, 'Error creating blog. Please try again!')
            return redirect('create-blog')
    else:
        form = BlogForm()
    context['form'] = form
    return render(request, 'create_blog.html', context)

def list_blogs(request):
    context = {}
    # for subscriber submission
    if request.method == "POST":
        email = request.POST['subscriberEmail']
        print(email, ">>>email")
        if Subscriber.objects.filter(email = email).exists() and email is not None:
            print("You have already subscribed")
            messages.info(request, "You have already subscribed")
            return redirect("list-blogs")
        else:
            subscriber = Subscriber.objects.create(email = email)
            send_mail(
                "EcoGuardian Subscription",
                "Thank you for subscribing, we will post you future news and blogs.",
                "priyaambaldhage@gmail.com",
                [subscriber.email],
                fail_silently=False,
            )
            print("You have subscribed successfully.")
            messages.success(request, 'You have subscribed successfully!')
            return redirect('subscription-success')
    else:
        all_blogs = Blog.objects.all()
        context['blogs'] = all_blogs
        return render(request, 'list_blog.html', context)

def subscription_success(request):
    return render(request, 'success_subcription.html')
```

```python
def detail_blog(request, blog_slug):
    context = {}
    blog = get_object_or_404(Blog, slug=blog_slug)
    if request.user.is_authenticated:
        if request.method == 'POST':
            form = LoggedUserCommentForm(request.POST)
            if form.is_valid():
                comment = form.save(commit=False)
                comment.blog = blog
                comment.user = request = request.user
                comment.save()
                messages.success(request, 'Thank you for commenting to this blog')
                return redirect('detail-blog', blog_slug = blog_slug)
            else:
                messages.error(request, 'Error creating comments. Please try again!')
                return redirect('detail-blog', blog_slug=blog_slug)
        else:
            form = LoggedUserCommentForm()
        context['loggeduser_form'] = form
    else:
        if request.method == 'POST':
            form = GuestUserCommentForm(request.POST)
            if form.is_valid():
                comment = form.save(commit=False)
                comment.full_name = form.cleaned_data['full_name']
                comment.email = form.cleaned_data['email']
                comment.blog = blog
                comment.save()
                messages.success(request, 'Thank you for commenting to this blog')
                return redirect('detail-blog', blog_slug = blog_slug)
            else:
                messages.error(request, 'Error creating comments. Please try again!')
                return redirect('detail-blog', blog_slug=blog_slug)
        else:
            form = GuestUserCommentForm()
        context['guestuser_form'] = form
    comments = Comment.objects.filter(blog__id = blog.id).order_by('-id')
    context['comments'] = comments
    context["blog"] = blog
    return render(request, 'detail_blog.html', context)
```

```python
def edit_blog(request, blog_slug):
    context = {}
    blog = get_object_or_404(Blog, slug=blog_slug)
    if request.method == 'POST':
        form = BlogForm(request.POST, instance=blog)
        if form.is_valid():
            blog = form.save(commit= False)
            blog.author = request.user
            blog.save()
            messages.success(request, 'Blog updated successfully')
            return redirect('list-blogs')
        else:
            messages.error(request, 'Error updating blog. Please try again!')
            return redirect('edit-blog', blog_slug=blog_slug)
    else:
        form = BlogForm(instance=blog)
    context['form'] = form
    return render(request, 'edit_blog.html', context)

def delete_blog(request, blog_slug):
    blog = get_object_or_404(Blog, slug=blog_slug)
    blog.delete()
    messages.success(request, 'Blog deleted successfully')
    return redirect('list-blogs')
```

This code provides a set of views for the blogging/ community forum platform. The key features are:

1. Create Blog: Allows users to create a new blog post. It handles both form submission and validation, saving the blog with the logged-in user as the author.
2. List Blogs: Displays all blogs and handles subscription to the blog via email. If a user subscribes, they receive an email confirmation.

3. Blog Details and Comments: Displays a specific blog's details, allowing logged-in users and guests to comment. Different forms are used based on whether the user is authenticated.
4. Edit Blog: Allows the author to edit an existing blog. It pre-populates the form with the current blog data and updates it upon submission.
5. Delete Blog: Enables the deletion of a blog, after which the user is redirected to the list of blogs.

Forms.py

```python
from django import forms
from blog.models import Blog, Comment
from django_summernote.widgets import SummernoteWidget, SummernoteInplaceWidget

class BlogForm(forms.ModelForm):
    class Meta:
        model = Blog
        fields = ['title', 'author', 'body']
        widgets = {
            'body' : SummernoteWidget(), # or use summernoteinplacewidget

        }

class LoggedUserCommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ['body']
        widgets = {
            'body' : forms.Textarea(attrs={'rows':3, 'placeholder':"Add your comment..."}),
        }

class GuestUserCommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ['full_name', 'email', 'body']
        widgets = {
            'body' : forms.Textarea(attrs={'rows':3, 'placeholder':"Add your comment...", 'required':True}),
            'email' : forms.EmailInput(attrs={'placeholder':"Enter your email address...", 'required':True}),
            'full_name' : forms.TextInput(attrs={'placeholder':"Add your full name..."}),

        }
```

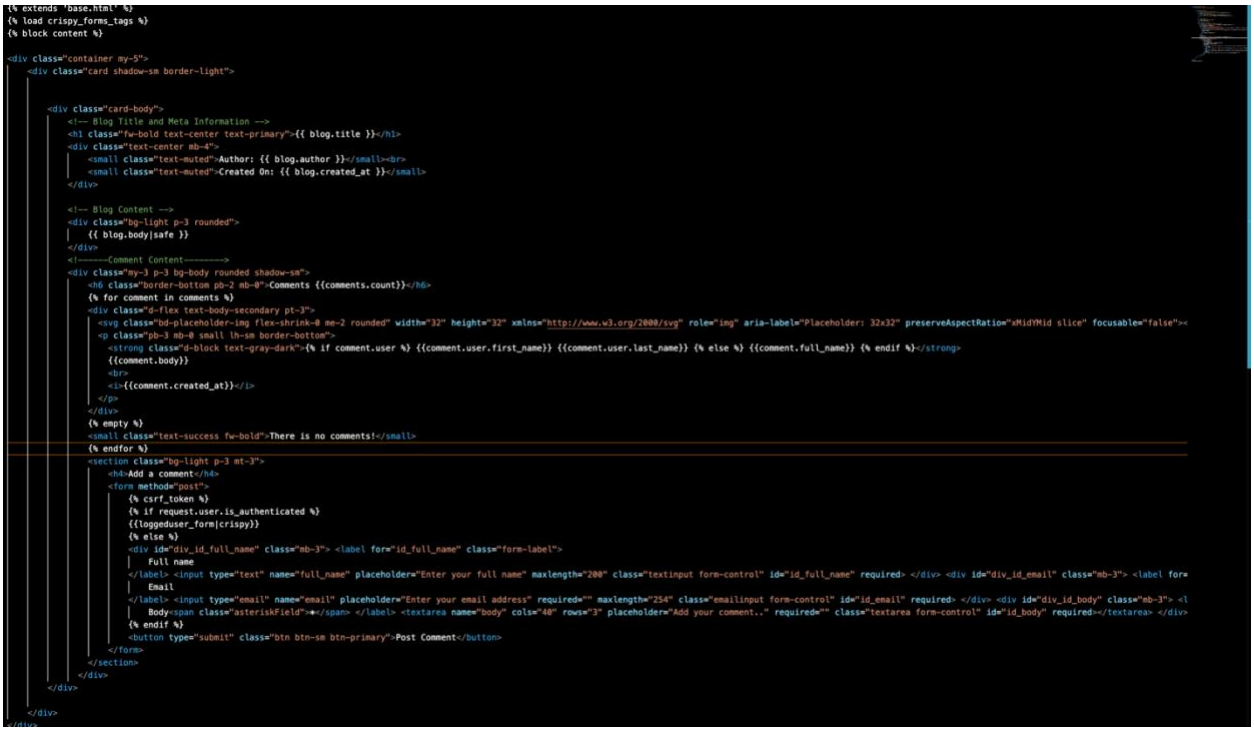
In the forms.py, the code defines forms for creating and editing blogs, as well as handling comments from both logged-in and guest users:

1. BlogForm: A form for creating and editing blog posts. It uses the SummernoteWidget for the body field to provide a rich-text editor.
2. LoggedUserCommentForm: A form for authenticated users to submit comments. It includes a simple text area for comment input, with a placeholder to guide users.
3. GuestUserCommentForm: A form for guest users to submit comments. It requires the user's full name, email, and comment body. Each field includes placeholders, and the email field is specified as required.

Create_blog.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div class="p-2">
        <h1 class="fw-bold text-center"> Create Blog</h1>
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{form|crispy}}
            <button type="submit" class="btn btn-lg btn-success">Save</button>
        </form>
    </div>
</div>
{% endblock content %}
```

Detail_blog.html



In this html, I have included, 3 parts

Blog Details: Where the blog's title, author, and creation date are displayed prominently. The blog content is rendered using the {{ blog.body|safe }} tag, ensuring safe HTML rendering.

Comments Section: It displays a list of existing comments for the blog, showing the commenter's name (logged-in user or guest) and comment body. A default message appears if no comments are available.

Comment Form: Logged-in users see a simple comment form powered by Django’s crispy forms. While guest users are required to enter their name, email, and comment body.

Edit_blog.html

```
{% extends 'base.html' %}
{% load crispy_forms_tags %}
{% block content %}
<div class="p-4 p-md-5 mb-4 rounded text-body-emphasis bg-body-secondary">
    <div class="p-2">
        <h1 class="fw-bold text-center"> Edit Blog</h1>
        <form method="post">
            {% csrf_token %}
            {{form|crispy}}
            <button type="submit" class="btn btn-lg btn-success">Update</button>
        </form>
    </div>
</div>
{% endblock content %}
```

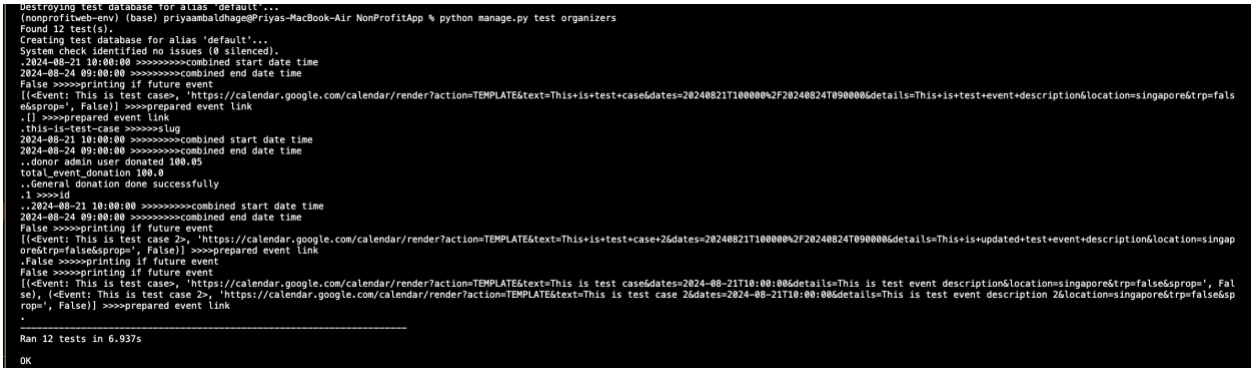
List_blog.html

```
{% extends 'base.html' %}
{% block content %}
<div class="container my-5">
    <div class="p-4 mb-4 rounded bg-light shadow-sm">
        <div class="text-end mb-4">
            {% if request.user.is_authenticated %}
            <a href="{% url 'create-blog' %}" class="btn btn-success btn-lg">Create Blog</a>
        {% endif %}
        </div>
        <!-- New Title and Subscribe Section -->
        <h2 class="fw-bold text-center mb-4">Inside Design: Stories and Interviews</h2>
        <p class="text-center">Subscribe to learn about new product features, the latest in technology, and updates.</p>
        <div class="text-center mb-5">
            <form class="form-control" method="post">
                {% csrf_token %}
            <input type="email" class="form-control d-inline-block w-auto"
            name = "subscriberEmail"
            placeholder="Enter your email">
            <button type=submit class="btn btn-primary">Subscribe</button>
            </form>
        </div>

        <h2 class="fw-bold text-center mb-4 ">Community Forum Page</h2>
        <div class="row">
            {% for blog in blogs %}
            <div class="col-sm-12 col-md-6 col-lg-4 mb-4">
                <div class="card border-light shadow-lg">
                    <div class="card-body">
                        <h4 class="card-title mb-2 text-dark">{{ blog.title }}</h4>
                        <h6 class="card-subtitle mb-3 text-muted">{{ blog.author }} | {{ blog.created_at }}</h6>
                        <p class="card-text">{{blog.body|safe|truncatechars:249}}</p>
                        <div class="d-flex justify-content-between align-items-center">
                            {% if request.user.is_authenticated and request.user.is_superuser %}
                            <div>
                                <a href="{% url 'edit-blog' blog.slug %}" class="btn btn-primary btn-sm me-2">Edit</a>
                                <a href="{% url 'delete-blog' blog.slug %}" class="btn btn-danger btn-sm">Delete</a>
                            </div>
                            {% endif %}
                            <a href="{% url 'detail-blog' blog.slug %}" class="btn btn-success btn-sm">Detail</a>
                        </div>
                    </div>
                </div>
            </div>
            {% empty %}
            <div class="text-center">
                <small class="fw-bold">There are no blogs yet</small>
            </div>
            {% endfor %}
        </div>
    </div>
</div>
{% endblock content %}
```

The key feature I have included in the html page is Create Blog Button where if the user is authenticated, a "Create Blog" button is shown for creating new blog posts. A subscription Section: which includes a subscription form for users to enter their email and receive updates about new features and technology. Blog Listings It displays a grid of blog posts with titles, authors, creation dates, and truncated content. Authenticated superusers have the option to edit or delete blog posts. Each blog post includes a "Detail" button linking to the full blog post view. A message is shown if there are no blogs available.

Code Testing: [xviii] [xix] [xx]

Organizers.tests.py [xxi]



```python
from django.test import TestCase
from django.urls import reverse
from django.contrib.auth.models import User
from django.contrib import messages
from django.contrib.messages import get_messages
from organizers.models import Category, Event, Sponser, Donation, Volunteer
from organizers.forms import EventForm, DonationForm, EventDonationForm
from datetime import datetime
from urllib.parse import urlencode
from organizers.views import create_google_calendar_url
from unittest.mock import patch

# Create your tests here.

class CreateEventTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.valid_data={
            'title': 'This is test case',
            'category': self.category.id,
            'date_from': '2024-08-21',
            'date_to': '2024-08-24',
            'time_from': '10:00:00',
            'time_to': '09:00:00',
            'description':'This is test event description',
            'venue': 'singapore',
            'is_expired':False,
            'is_active':True,
            'is_volunteer_required':False,
            'volunteers':{self.volunteer.id},
            'sponsers': {self.sponser.id},
            'volunteers_capacity': 1,
            'terms': 'This is test terms'
        }

    def test_create_event_with_valid_data(self):
        response = self.client.post(reverse('create_events'), data=self.valid_data, follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertRedirects(response, reverse('all-events'))
        self.assertTrue(Event.objects.filter(title='This is test case').exists())

    def test_create_event_with_invalid_data(self):
        invalid_data = self.valid_data.copy()
        invalid_data['title'] = ''
        invalid_data['date_from'] = ''
        response = self.client.post(reverse('create_events'), data=invalid_data, follow=True)
        self.assertEqual(response.status_code, 200)
        #self.assertTrue(Event.objects.filter(title='This is test event').exists())
        self.assertRedirects(response, reverse('create_events'))
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages),0)
        self.assertEqual(str(messages[0]), 'Error creating event. Please try again!')
```

```python
class EditEventTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.event=Event.objects.create(
            title = 'This is test case',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )
        self.valid_data={
            'title': 'This is test case 2',
            'category': self.category.id,
            'date_from': '2024-08-21',
            'date_to': '2024-08-24',
            'time_from': '10:00:00',
            'time_to': '09:00:00',
            'description':'This is updated test event description',
            'venue': 'singapore',
            'is_expired':False,
            'is_active':True,
            'is_volunteer_required':False,
            'volunteers':{self.volunteer.id},
            'sponsers': {self.sponser.id},
            'volunteers_capacity': 1,
            'terms': 'This is test terms'
        }

    def test_edit_event_get_request(self):
        self.event.volunteers.set([self.volunteer])
        self.event.sponsers.set([self.sponser])

        print(self.event.id,">>>>id")
        response = self.client.get(reverse('edit_events', args=[self.event.id]))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'organizers/edit_event.html')

        #check the form whether getting the data or not
        form = response.context['form']
        self.assertIsInstance(form, EventForm)
        self.assertEqual(form.instance, self.event)
        self.assertEqual(form.initial['title'], self.event.title)

        self.assertIn(self.volunteer, form.instance.volunteers.all())
        self.assertIn(self.sponser, form.instance.sponsers.all())

    def test_edit_event_with_valid_data(self):
        response = self.client.post(reverse('edit_events', args=[self.event.id]), data=self.valid_data, follow=True)
        #self.assertEqual(response.status_code, 200)
        self.assertRedirects(response, reverse('all-events'))
        self.event.refresh_from_db()
        self.assertTrue(self.event.title, 'This is test case 2')
        self.assertTrue(self.event.description, 'This is updated test event description')
```

```python
    def test_edit_event_with_invalid_data(self):
        invalid_data = self.valid_data.copy()
        invalid_data['title'] = ''
        invalid_data['description'] = ''
        response = self.client.post(reverse('edit_events',args=[self.event.id]), data=invalid_data, follow=True)
        self.assertEqual(response.status_code, 200)
        #self.assertTrue(Event.objects.filter(title='This is test event').exists())
        self.assertRedirects(response, reverse('edit_events', args=[self.event.id]))
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages),0)
        self.assertEqual(str(messages[0]), 'Error updating event. Please try again!')


class DeleteEventTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.event=Event.objects.create(
            title = 'This is test case',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )

    def test_delete_event(self):
        self.assertTrue(Event.objects.filter(id=self.event.id).exists)
        response = self.client.post(reverse('delete_events', args=[self.event.id]), follow=True)
        self.assertFalse(Event.objects.filter(title='This is test case'). exists())
        self.assertRedirects(response, reverse('all-events'))

class DetailEventTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.event=Event.objects.create(
            title = 'This is test case',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description',
            banner= 'image.jpg',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )

    def test_detail_event(self):
        print(self.event.slug,">>>>>>slug")
        response = self.client.get(reverse('event_details', args=[self.event.slug]), follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'organizers/event_detail.html')
        self.assertEqual(response.context['event_detail'], self.event)

        combine_start_date_time = datetime.strptime(f'{self.event.date_from} {self.event.time_from}', '%Y-%m-%d %H:%M:%S')
        combine_end_date_time = datetime.strptime(f'{self.event.date_to} {self.event.time_to}', '%Y-%m-%d %H:%M:%S')

        # format the datetime obj
        start_formatted_dattime = combine_start_date_time.strftime('%Y%m%dT%H%M%S')
        end_formatted_dattime = combine_end_date_time.strftime('%Y%m%dT%H%M%S')

        params = {
        'action':'TEMPLATE',
        'text':self.event.title,
        'dates':f'{start_formatted_dattime}/{end_formatted_dattime}',
        'details': self.event.description,
        'location': self.event.venue,
        'trp':'false',
        'sprop':''
        }
        expected_google_calender_url = f'https://calendar.google.com/calendar/render?{urlencode(params)}'
        self.assertEqual(response.context['google_calender_url'], expected_google_calender_url)
```

```python
class ListAllEventsTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.event1=Event.objects.create(
            title = 'This is test case',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description',
            banner= 'image.jpg',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )
        self.event2=Event.objects.create(
            title = 'This is test case 2',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description 2',
            banner= 'image.jpg',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )
    @patch('organizers.views.create_google_calendar_url')
    def test_all_events(self, mock_create_google_calander_url):
        mock_create_google_calander_url.side_effect = lambda event: f'https://calendar.google.com/calendar/render?action=TEMPLATE&text={event.title}&dates={event.date_from}T{event.time_from}&details={event.description}&locatio

        response = self.client.get(reverse('all-events'), follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'organizers/all_events.html')
        self.assertEqual(len(response.context['all_events']), 2)

        # check the event and google calander url for the event1
        event1, google_calander_url1, is_future_event1  = response.context['all_events'][0]
        self.assertEqual(event1, self.event1)
        self.assertEqual(google_calander_url1, mock_create_google_calander_url(self.event1))
        self.assertFalse(is_future_event1)

        # check the event and google calander url for the event1
        event2, google_calander_url2, is_future_event2  = response.context['all_events'][1]
        self.assertEqual(event2, self.event2)
        self.assertEqual(google_calander_url2, mock_create_google_calander_url(self.event2))
        self.assertFalse(is_future_event2)


class DonationTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.volunteer = Volunteer.objects.create(user = self.user, dob_month=4, dob_day=1, dob_year=2021, phone='81234567', full_address='test_address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email='sponser@gmail.com', contact='91234567')

        self.event=Event.objects.create(
            title = 'This is test case',
            category = self.category,
            date_from = '2024-08-21',
            date_to = '2024-08-24',
            time_from = '10:00:00',
            time_to = '09:00:00',
            description = 'This is test event description',
            banner= 'image.jpg',
            venue = 'singapore',
            is_expired = False,
            is_active = True,
            is_volunteer_required = False,
            volunteers_capacity = 1,
            terms = 'This is test terms'
        )

    def test_donation_get_request(self):
        response = self.client.get(reverse('donation'), follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'donation/donate.html')
        self.assertIn('get_all_donors', response.context)
        self.assertIn('total_general_donation', response.context)
        self.assertIn('events', response.context)
        self.assertIn('total_event_donation', response.context)
        self.assertIn('fund_raising_events', response.context)

    def test_donation_post_request(self):
        data = {
            'inputDonorName': 'user1',
            'inputAmount': 20.00
        }
        response = self.client.post(reverse('donation'), data=data, follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertRedirects(response, reverse('donation'))
        self.assertTrue(Donation.objects.filter(donated_name='user1', amount=20.00, if_donated_for_event=False).exists())
```

```python
class DonateToEventTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000', first_name='admin', last_name='user')
        self.client.login(username='admin', password='admin@000')

        self.category = Category.objects.create(name='test category')
        self.volunteer = Volunteer.objects.create(user=self.user, dob_month=4, dob_day=1, dob_year=2024, phone='545454545', full_address='test address')
        self.sponser = Sponser.objects.create(sponser='Test sponser', email="sponser@gmail.com", contact='54545454')

        self.event=Event.objects.create(
            title='This is test event',
            category=self.category,
            date_from= '2024-08-08',
            date_to= '2024-08-24',
            time_from= '10:00:00',
            time_to= '18:00:00',
            description= 'This is test event description',
            banner='image.jpg',
            venue= 'Bhilai',
            is_expired=False,
            is_active=True,
            is_volunteer_required=True,
            volunteers_capacity=1,
            terms='This is test terms'
        )

    def test_donation_to_event_get_request(self):
        response = self.client.get(reverse('donation_to_event', args=[self.event.slug]), follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'donation/donate_to_event.html')
        self.assertIn('event', response.context)
        self.assertEqual(response.context['event'], self.event)


    def test_donation_to_event_post_request_authenticate(self):
        data = {
            'inputAmountEvent':100.05
        }
        full_name = self.user.first_name + ' ' + self.user.last_name
        response = self.client.post(reverse('donation_to_event', args=[self.event.slug]), data=data, follow=True)
        self.assertRedirects(response, reverse('donation'))
        self.assertTrue(Donation.objects.filter(donated_name=full_name, amount=100.05, if_donated_for_event=True, event=self.event).exists())

        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'Thank you for donating to this event')
```

The above code ensures the functionality of creating, editing, deleting, viewing details, and donating to events in the project. The test cases use Django's TestCase class, allowing database testing without affecting actual data.

1. Setup Methods: The setUp method in each class initializes necessary objects such as a user, category, volunteer, sponsor, and event. It also logs in the test user using client.login, ensuring they have access to the protected views.
2. Create Event Test: This class tests event creation with both valid and invalid data. In the test_create_event_with_valid_data, a POST request is made with valid data to ensure that the event is created and the user is redirected to the events list page. In contrast, the test_create_event_with_invalid_data checks whether the system correctly handles invalid input by displaying an error message and preventing event creation.
3. Edit Event Test: This section tests the ability to retrieve and update event details. The test_edit_event_get_request checks if the event's edit form is correctly populated with existing data. The test_edit_event_with_valid_data verifies successful updates with valid data, while the invalid data test ensures that errors are handled gracefully.
4. Delete Event Test: This part tests event deletion. It confirms that after a DELETE request, the event no longer exists in the database, and the user is redirected to the event listing page.
5. Detail Event Test: This class checks if the event detail page loads correctly with the expected data, including the generation of a Google Calendar link. The start and end times are formatted and passed as query parameters to create a calendar event URL.
6. List All Events Test: It verifies that the page displaying all events works correctly. It mocks the Google Calendar URL generation and checks if the correct events and links are returned to the page context.
7. Donation Test: This class tests the donation system. The test_donation_get_request ensures that the donation page loads correctly, while the test_donation_post_request checks if a donation is successfully saved when valid data is submitted.

8. Donate to Event Test: It tests the donation functionality for specific events. The GET request ensures that the event-specific donation page loads properly, while the POST request simulates donating to an event. It ensures the donation is saved and a thank-you message is displayed.

Report.tests.py

```python
class SubmitReportTest(TestCase):
    def setUp(self):
        self.report = Report.objects.create(
            report_type='Pet Violence',
            description='This is a test report.',
            location='Test Location',
            contact_email='test@example.com',
            contact_phone='1234567890'
        )

        self.report_status = ReportStatus.objects.create(
            report=self.report,
            is_completed=False
        )

    def test_report_creation(self):
        self.assertEqual(self.report.report_type, 'Pet Violence')
        self.assertEqual(self.report.description, 'This is a test report.')
        self.assertEqual(self.report.location, 'Test Location')
        self.assertEqual(self.report.contact_email, 'test@example.com')
        self.assertEqual(self.report.contact_phone, '1234567890')

    def test_submit_report_view(self):
        response = self.client.post(reverse('submit-report'), {
            'report_type': 'Pet Violence',
            'description': 'This is a test report.',
            'location': 'Test Location',
            'contact_email': 'test@example.com',
            'contact_phone': '1234567890'
        }, follow=True)
        report = Report.objects.last()
        self.assertEqual(report.report_type, 'Pet Violence')
        self.assertEqual(report.description, 'This is a test report.')

        self.assertEqual(len(mail.outbox), 1)
        self.assertEqual(mail.outbox[0].subject, 'EcoGuardian Report Submitted')
        self.assertIn('test@example.com', mail.outbox[0].to)
        self.assertRedirects(response, reverse('report-thank-you'))

class ReportListViewTest(TestCase):
    def setUp(self):
        self.report1 = Report.objects.create(
            report_type='Pet Violence',
            description='This is a test report.',
            location='Test Location',
            contact_email='test@example.com',
            contact_phone='1234567890'
        )
        self.report2 = Report.objects.create(
            report_type='Animal Abuse',
            description='This is another test report.',
            location='Test Location 2',
            contact_email='test2@example.com',
            contact_phone='0987654321'
        )

        self.report_status1 = ReportStatus.objects.create(
            report=self.report1,
            is_completed=False
        )
        self.report_status2 = ReportStatus.objects.create(
            report=self.report2,
            is_completed=True
        )
```

```
    def test_reports_status_list_view(self):
        response = self.client.get(reverse('report-list'))
        self.assertEqual(response.status_code, 200)
        self.assertEqual(ReportStatus.objects.count(), 2)


class ReportActionTest(TestCase):
    def setUp(self):
        self.report = Report.objects.create(
            report_type='Pet Violence',
            description='This is a test report.',
            location='Test Location',
            contact_email='test@example.com',
            contact_phone='1234567890'
        )
        self.report_status = ReportStatus.objects.create(
            report=self.report,
            is_completed=False
        )

    def test_report_completed_status(self):
        response = self.client.get(reverse('report-completed', args=[self.report_status.id]))
        self.report_status.refresh_from_db()
        self.assertTrue(self.report_status.is_completed)

        self.assertEqual(len(mail.outbox), 1)
        self.assertEqual(mail.outbox[0].subject, 'EcoGuardian Report Completed')
        self.assertIn(self.report.contact_email, mail.outbox[0].to)
        self.assertRedirects(response, reverse('report-list'))
```

SubmitReportTest focuses on testing the report submission functionality. It begins by setting up a Report and ReportStatus instance. The test_report_creation method ensures that a report is correctly created with the expected details. The test_submit_report_view method verifies that when a report is submitted through the view, a new report is created, an email notification is sent, and the user is redirected to a thank-you page.

ReportListViewTest evaluates the functionality of the report listing view. It sets up two reports with different statuses. The test_reports_status_list_view method checks that the report list view responds with a status code of 200 and confirms that the number of ReportStatus records matches the expected count.

ReportActionTest tests the functionality for marking reports as completed. After setting up a report and its status, the test_report_completed_status method ensures that accessing the completion view successfully updates the report status to completed, sends an appropriate email notification, and redirects the user to the report list view.

```
(nonprofitweb-env) (base) priyaambaldhage@Priyas-MacBook-Air NonProfitApp % python manage.py test report
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
----------------------------------------------------------------------
Ran 4 tests in 0.040s

OK
Destroying test database for alias 'default'
```

Recycle tests.py

```python
class RecycleRequestTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')


    def test_recycle_request_create_authenticated(self):
        self.recycle1 = {
            'id': 1,
            'user':self.user,
            'title':'recycle test title',
            'first_name':'',
            'last_name' : '',
            'email': '',
            'contact':'',
            'full_address':'address1',
            'recycle_items':'paper, plastic'}
        response = self.client.post(reverse('create-recycle-request'), data=self.recycle1)
        self.assertEqual(response.status_code, 302)
        self.assertRedirects(response, reverse('create-recycle-request'))
        self.assertEqual(RecycleRequest.objects.count(),1)
        self.assertTrue(RecycleRequest.objects.filter(user=self.user).exists())


class RecycleRequestViewTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')

        self.guest_user_data = {
            'title':'recycle glass',
            'first_name':'test',
            'last_name' : 'user',
            'email': 'test@gmail.com',
            'contact':'4545454',
            'full_address':'address2',
            'recycle_items':'glass1, glass2'
        }
        self.logged_user_data = self.guest_user_data.copy()
        self.logged_user_data.update({
            'user':self.user,
            'title': 'recycle metal'
        })

    def test_create_recycle_request_authenticated_user_post(self):
        response = self.client.post(reverse('create-recycle-request'), data=self.logged_user_data, follow=True)
        self.assertRedirects(response, reverse('create-recycle-request'))
        self.assertTrue(RecycleRequest.objects.filter(title='recycle metal').exists())
        self.assertEqual(RecycleRequest.objects.get(title='recycle metal').user, self.user)


    def test_create_recycle_request_guest_user_post(self):
        response = self.client.post(reverse('create-recycle-request'), data=self.guest_user_data, follow=True)
        self.assertRedirects(response, reverse('create-recycle-request'))
        self.assertTrue(RecycleRequest.objects.filter(title='recycle glass').exists())
        self.assertFalse(RecycleRequest.objects.get(title='recycle glass').user == '')


class RecycleRequestListViewTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.recycle_request1 = RecycleRequest.objects.create(
            user=self.user,
            title='recycle paper',
            full_address='address1',
            recycle_items='paper'
        )
        self.recycle_request2 = RecycleRequest.objects.create(
            user=self.user,
            title='recycle plastic',
            full_address='address1',
            recycle_items='plastic'
        )
        self.status1 = RecycleRequestStatus.objects.create(
            request=self.recycle_request1
        )
        self.status2 = RecycleRequestStatus.objects.create(
            request=self.recycle_request2
        )

    def test_recycle_request_list_view(self):
        response = self.client.get(reverse('recycle-request-list'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'recycle_list.html')
        all_recycle_requests = response.context['all_recycle_requests']
        self.assertEqual(all_recycle_requests.count(), 2)
        self.assertEqual(all_recycle_requests[0].request.title, 'recycle plastic')
        self.assertEqual(all_recycle_requests[1].request.title, 'recycle paper')


    def test_recycle_request_empty(self):
        RecycleRequest.objects.all().delete()
        response = self.client.get(reverse('recycle-request-list'))
        self.assertEqual(response.status_code, 200)
        self.assertEqual(len(response.context['all_recycle_requests']), 0)
```

RecycleRequestTest validates the functionality of creating a recycling request. In the setUp method, an authenticated user is created and logged in. The test_recycle_request_create_authenticated method checks if an authenticated user can successfully submit a recycling request. It ensures the request is redirected to the appropriate page, is saved in the database, and is associated with the logged-in user.

RecycleRequestViewTest focuses on testing the creation of recycling requests for both authenticated and guest users. In setUp, user and guest data are prepared.
The test_create_recycle_request_authenticated_user_post method verifies that an authenticated user can submit a request, which is saved and associated correctly with the user.
Similarly, test_create_recycle_request_guest_user_post ensures that guest users can submit a request as well, with proper checks to confirm the request is saved and not associated with a user.

RecycleRequestListViewTest tests the functionality of listing recycling requests. After setting up multiple requests and their statuses, the test_recycle_request_list_view method verifies that the list view displays the correct number of requests and orders them properly. The test_recycle_request_empty method ensures that the list view correctly shows no requests when the database is empty.

```
(nonprofitweb-env) (base) priyaambaldhage@Priyas-MacBook-Air NonProfitApp % python manage.py test recycle
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
<QuerySet []> *****
.<QuerySet [<RecycleRequestStatus: None - False>, <RecycleRequestStatus: None - False>]> *****
....
----------------------------------------------------------------------
Ran 5 tests in 2.929s

OK
```

New Tests.py

```
class NewsCreatTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.valid_data={
            'title':'This is test news',
            'category':self.category.id,
            'body': 'This is test news description',
            'author':self.user.id
            }

    def test_create_news_with_valid_data(self):
        response = self.client.post(reverse('create-news'), self.valid_data)
        self.assertEqual(response.status_code, 302)
        self.assertEqual(News.objects.count(), 1)
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'News created successfully')
        self.assertRedirects(response, reverse('list-news'))

    def test_create_news_with_invalid_data(self):
        invalid_data = self.valid_data.copy()
        invalid_data['title'] = ''
        response = self.client.post(reverse('create-news'), invalid_data)
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'Error creating news')
        self.assertRedirects(response, reverse('create-news'))


class NewsListTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.news = News.objects.create(title='This is test news', category=self.category, body='This is test news description', author=self.user)

    def test_list_news_view(self):
        response = self.client.get(reverse('list-news'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, self.news.title)
        self.assertTemplateUsed(response, 'list_news.html')


class NewsDetailTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.news = News.objects.create(title='This is test news', category=self.category, body='This is test news description', author=self.user)
```

```python
    def test_news_detail_view(self):
        response = self.client.get(reverse('detail-news', args=[self.news.slug]))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, self.news.title)
        self.assertContains(response, self.news.body)
        self.assertTemplateUsed(response, 'detail_news.html')


class NewsEditTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.news = News.objects.create(title='This is test news', category=self.category, body='This is test news description', author=self.user)
        self.valid_data={
            'title':'This is test news updated',
            'category':self.category.id,
            'body': 'This is test news description updated',
            'author':self.user.id
        }

    def test_edit_news_with_valid_data(self):
        response = self.client.post(reverse('edit-news', args=[self.news.slug]), self.valid_data)
        self.assertEqual(response.status_code, 302)
        self.assertEqual(News.objects.count(), 1)
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'News updated successfully')
        self.assertRedirects(response, reverse('list-news'))


    def test_edit_news_with_invalid_data(self):
        invalid_data = self.valid_data.copy()
        invalid_data['title'] = ''
        response = self.client.post(reverse('edit-news', args=[self.news.slug]), data=invalid_data)
        self.assertEqual(response.status_code, 302)
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'Error updating news')
        self.assertRedirects(response, reverse('edit-news', args=[self.news.slug]))


class NewsDeleteTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')
        self.client.login(username='admin', password='admin@000')
        self.category = Category.objects.create(name='animal rescue')
        self.news = News.objects.create(title='This is test news', category=self.category, body='This is test news description', author=self.user)

    def test_delete_news(self):
        response = self.client.post(reverse('delete-news', args=[self.news.slug]))
        self.assertEqual(response.status_code, 302)
        self.assertEqual(News.objects.count(), 0)
        messages = list(get_messages(response.wsgi_request))
        self.assertGreater(len(messages), 0)
        self.assertEqual(str(messages[0]), 'News deleted successfully')
        self.assertRedirects(response, reverse('list-news'))
```

NewsCreateTest: In the setUp method, a user and a category are created, and valid form data is set up. The test_create_news_with_valid_data ensures that a user can successfully create a news article, receives a success message, and is redirected to the news list page.
The test_create_news_with_invalid_data tests form validation by submitting an empty title, ensuring that the news article is not created, an error message is displayed, and the user is redirected back to the creation form.

NewsListTest: This test validates that news articles are correctly listed. After creating a news article in setUp, the test_list_news_view ensures that the list view returns a successful response and displays the news article correctly, using the appropriate template.

NewsDetailTest:This test verifies the detail view of a news article.
The test_news_detail_view method checks that the news article's details are displayed correctly (title and body) and that the appropriate template is used for rendering.

NewsEditTest: In this test, both valid and invalid edit operations are tested.
test_edit_news_with_valid_data ensures that an authenticated user can edit a news article and receive a success message upon completion. test_edit_news_with_invalid_data checks for form validation by submitting an invalid title (empty), ensuring that the error is handled correctly and the user is redirected back with an error message.

NewsDeleteTest: This test checks if a news article can be deleted successfully.
The test_delete_news method verifies that after deletion, the article no longer exists in the database, a success message is displayed, and the user is redirected to the news list.

```
(nonprofitweb-env) (base) priyaambaldhage@Priyas-MacBook-Air NonProfitApp % python manage.py test news
Found 7 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.......
----------------------------------------------------------------------
Ran 7 tests in 4.051s

OK
```

Volunteer Test

.js

```
class LoginUserTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='admin', password='admin@000')

    def test_login_user_valid_credentials(self):
        data = {
            'username': 'admin',
            'password': 'admin@000'
        }
        response = self.client.post(reverse('login-user'), data=data, follow=True)
        user = authenticate(username='admin', password='admin@000')
        self.assertTrue(user)
        self.assertRedirects(response, reverse('home'))

    def test_login_user_invalid_credentials(self):
        data = {
            'username': 'abc',
            'password': 'abc@0000'
        }
        response = self.client.post(reverse('login-user'), data=data, follow=True)
        user = authenticate(username='abc', password='abc@000')
        self.assertFalse(user)

    def test_login_user_get_request(self):
        response = self.client.get(reverse('login-user'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'login_user.html')
        self.assertIn('user_form', response.context)


class RegisterUserTest(TestCase):
    def setUp(self):
        self.user_data = {
            'username': 'testuser',
            'email': 'test@gmail.com',
            'first_name': 'test',
            'last_name': 'user',
            'password': 'test@000'
        }
        self.volunteer_data = {
            'gender':'male',
            'dob_day':5,
            'dob_month':5,
            'dob_year':2000,
            'phone':'4545454',
            'full_address':'test address',
            'profile_pic':'image.jpg'
        }
```

```python
def test_register_user_valid_data(self):
    data = {**self.user_data, **self.volunteer_data}
    response = self.client.post(reverse('register-user'), data=data, follow=True)

    # check user created
    user = User.objects.get(username='testuser')
    self.assertTrue(user)

    # check volunteer created
    volunteer = Volunteer.objects.get(user=user)
    self.assertEqual(volunteer.phone, '4545454')

    self.assertRedirects(response, reverse('login-user'))


def test_register_user_invalid_data(self):
    invalid_data = {**self.user_data, 'username':''} # passing username empty means it's invalid
    data = {**invalid_data, **self.volunteer_data}

    response = self.client.post(reverse('register-user'), data = data, follow=True)
    self.assertFalse(User.objects.filter(email='test@gmail.com').exists())
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'register_user.html')


def test_register_user_get_request(self):
    response = self.client.get(reverse('register-user'))
    self.assertEqual(response.status_code, 200)
    self.assertTemplateUsed(response, 'register_user.html')
    self.assertIn('user_form', response.context)
    self.assertIn('volunteer_form', response.context)



ass DeleteUserTest(TestCase):
def setUp(self):
    self.user = get_user_model().objects.create_user(username='testuser', email='test@gmail.com', password='test@000')
    self.client.login(username='testuser', password='test@000')

def test_logout_user(self):
    # first check if user is logged in or not
    response = self.client.get(reverse('home'))
    self.assertTrue(response.wsgi_request.user.is_authenticated)

    # now logout the user
    response = self.client.get(reverse('logout_user'))
    self.assertFalse(response.wsgi_request.user.is_authenticated)

    self.assertRedirects(response, reverse('home'))
```

```
(nonprofitweb-env) (base) priyaambaldhage@Priyas-MacBook-Air NonProfitApp % python manage.py test volunteer
Found 7 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.......
----------------------------------------------------------------------
Ran 7 tests in 2.952s

OK
```

LoginUserTest:

test_login_user_valid_credentials: This test ensures that a user with valid credentials can log in successfully. After logging in, the user is redirected to the home page.

test_login_user_invalid_credentials: This test verifies that a user with incorrect credentials cannot log in. The authenticate method returns False, confirming that login failed.

test_login_user_get_request: Ensures that a GET request to the login page returns the correct form and uses the login_user.htmltemplate. The form is available in the context.

RegisterUserTest:

test_register_user_valid_data: This test ensures that when a valid registration form is submitted, both a User and a corresponding Volunteer profile are created successfully. After registration, the user is redirected to the login page.

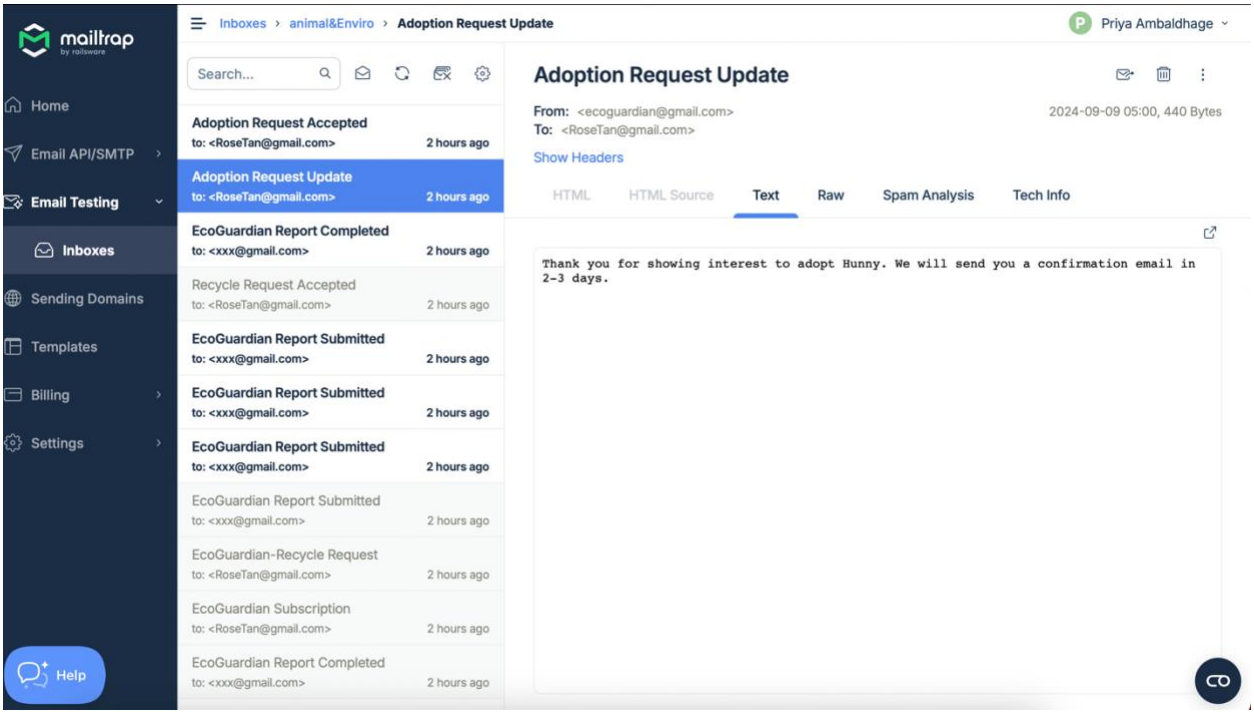
test_register_user_invalid_data: This test verifies that invalid registration data (e.g., an empty username) prevents user creation. It ensures that the registration form is redisplayed with validation errors, and no User or Volunteer is created.

test_register_user_get_request: Ensures that the GET request to the registration page returns the correct form (both UserForm and VolunteerForm) and renders the register_user.html template.

DeleteUserTest:

test_logout_user: This test first checks if the user is logged in. Then, after calling the logout view, it verifies that the user is successfully logged out and redirected to the home page.

Email Notifications using MailTrap 3rd party email [xxii]



(3691 words)

Evaluation:

Google Forms Survey:

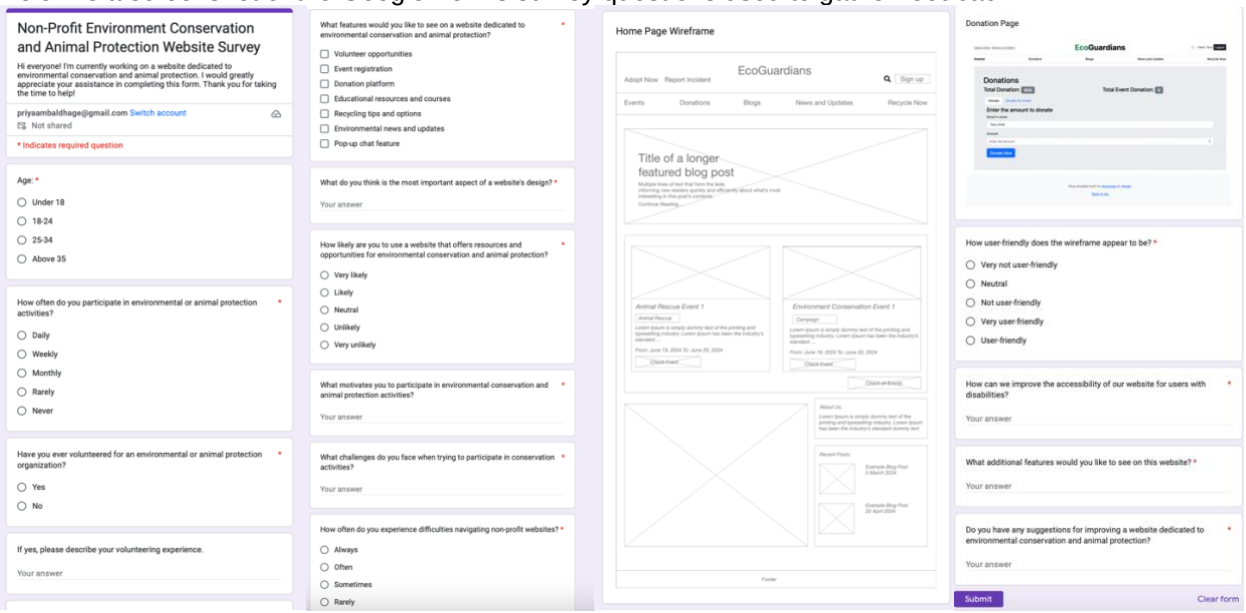
The evaluation of the website focused on gathering user feedback through a Google Forms survey. This approach provided insights into user perceptions and identified areas for improvement based on wireframe designs. A comprehensive Google Forms survey was distributed to gather feedback on the wireframe design of the website. The survey included questions about the ease of navigation, overall design aesthetics, user-friendliness, and desired features.

The survey responses provided valuable insights into the wireframe's effectiveness. Key findings include:

1. User-Friendliness: Most respondents found the wireframe to be user-friendly. They appreciated the clear layout and the intuitive placement of navigation elements.
2. Design Aesthetics: The majority of users liked the visual design of the wireframe, describing it as clean and appealing. However, a few respondents suggested enhancements such as incorporating more vibrant colours and images to make the site more engaging.
3. Navigation Ease: Users reported that the navigation was straightforward, with important sections like volunteer opportunities and donation options easy to locate. Some respondents suggested adding a fixed navigation bar for better accessibility.
4. Desired Features: Feedback highlighted a desire for additional features, including: Interactive elements such as community forums and discussion boards, more detailed event calendars and notification systems, enhanced search functionality to quickly find specific information.
5. Accessibility: Respondents emphasized the importance of accessibility features, recommending the inclusion of adjustable text sizes, voice command capabilities, and screen reader compatibility to ensure the site is usable by individuals with disabilities.

   Link:https://docs.google.com/forms/d/e/1FAIpQLScZTKEScb1JlEcqQAVf3gyC2lc6EfKP1pZ0rthHYfmHY_6ZOg/viewform?usp=sf_link

Below is a screenshot of the Google Forms survey questions used to gather feedback:



Do you have any suggestions for improving a website dedicated to environmental conservation and animal protection?

14 responses

- Nil
- NA
- Need to improve website design
- To improve the website, consider adding a chat support feature for instant assistance, more interactive content like quizzes, and regular updates on ongoing projects and their impacts.
- Ensure regular updates to keep the content fresh, add more visual content like videos and infographics, and offer mobile app support for on-the-go access.
- Include more interactive content, regularly update the site with new information, and offer a mobile-friendly version.

What additional features would you like to see on this website?

7 responses

- Additional features could include a blog with success stories, a newsletter subscription, and interactive maps of conservation areas.
- A community forum page for discussion
- A section for success stories, virtual tours of conservation sites, and a rewards program for active participants.
- N/A
- videos of past events and effects

What challenges do you face when trying to participate in conservation activities?

14 responses

- Inaccessibility
- Short of money
- One of the main challenges is finding time to participate amidst a busy schedule. Additionally, sometimes it is difficult to find local opportunities.
- Limited availability of local activities and lack of information on how to get involved.
- Finding time and opportunities that are accessible for someone my age.
- No readily available redo
- lack of opportunities
- My time is limited. Also, most difference can be made by a large group working together: so organizing is a challenge.

How can we improve the accessibility of our website for users with disabilities?

7 responses

- Improving accessibility could include adding alt text for images, ensuring keyboard navigation is possible, and providing text-to-speech functionality.
- Incorporate features like voice commands, adjustable text size, and screen reader compatibility.
- Adding features like keyboard navigation
- N/A
- Decreading density of information on the first page
- Add alt text, reduce the number of clickable items, add set suggestions for donation. Basically, reduce the number of clicks to a minimum, and ensure all images are accompanied by alt text. Do not over fill a page with images and text, keep it simple.
- Dyslexic typeface

User Persona 1 based on the answers from the survey:

Name: Jason Loh
Age: 18
Occupation: Polytechnic Student
Location: Singapore

Background: Jason is a 19-year-old student studying environmental science at a polytechnic in Singapore. He is deeply interested in sustainability and conservation, often participating in related activities organized by his institution.

Goals:
- To discover and engage in local and global volunteer opportunities.
- To access reliable and educational resources on environmental issues.
- To contribute to conservation projects through volunteering and donations.

Behaviours:
- Regularly participates in environmental conservation activities through his school.
- Actively looks for new ways to contribute to environmental protection.
- Uses digital platforms to stay informed about the latest in conservation efforts.

Pain Points:
- Finding volunteer opportunities that align with his academic schedule.
- Accessing comprehensive and up-to-date information on environmental initiatives.
- Encountering difficulties with complex website navigation.

Motivation:
- Finding volunteer opportunities that align with his academic schedule.
- Accessing comprehensive and up-to-date information on environmental initiatives.
- Encountering difficulties with complex website navigation.

Preferred Features on Website:
- Interactive maps showcasing volunteer opportunities.
- Real-time updates on conservation projects.
- An intuitive volunteer portal for easy sign-ups.
- Online donation options to support various initiatives.
- Access to webinars and educational content related to environmental science.

Conclusion: Jason Lim embodies the tech-savvy, environmentally conscious young adults who are enthusiastic about participating in and contributing to conservation efforts. By addressing his needs and preferences, the website can effectively engage similar users, enhancing overall user satisfaction and participation.

User Persona 2 based on the answers from the survey:

Name: Sarah Johnson
Age: 34
Occupation: Employee
Location: Singapore

Background: Sarah is an engaged community member who volunteers her time to help organize and manage local events. She often works with volunteers and sponsors to ensure the success of these events.

Goals:
- Efficiently manage and track various community events.
- Coordinate with volunteers and sponsors to facilitate their involvement.
- Keep event information updated and accessible for all stakeholders.

Pain Points:
- Juggling multiple event details and updates together with job.
- Tracking volunteer and sponsor participation effectively.
- Managing donations related to specific events.

Motivation:
- Finding volunteer opportunities that align with his academic schedule.
- Accessing comprehensive and up-to-date information on environmental initiatives.
- Encountering difficulties with complex website navigation.

Preferred Features on Website:
- Intuitive event creation and management tools.
- Easy interfaces for managing volunteers and sponsors.
- Clear donation tracking and reporting functionalities.

Conclusion: Understanding the needs and goals of user personas like Sarah Johnson helps tailor the non-profit website to better support its users. By providing intuitive tools for event management, volunteer coordination, sponsorship tracking, and donation handling, the site can effectively address the challenges faced by individuals in roles like Sarah's. This ensures that the website enhances efficiency and engagement for all users involved in organizing and managing community events.

User Testing:

User 1:

Name: Sujata
Age: 45
Occupation: Working Women
Location: Singapore .

Pain Points of the Webpage:

- It’s challenging to determine which pets are still available for adoption in the “Adopt Now” section.

Preferred Features on the Website:

- An interactive display for volunteer opportunities.
- An intuitive portal for seamless volunteer sign-ups.
- Options for online donations to support different initiatives.
- Access to webinars and educational resources focused on environmental science.

User 2:

Name: Tanu
Age: 20
Occupation: Student
Location: Singapore .

Pain Points:

- The website felt somewhat uninteresting despite being user-friendly.

Preferred Features:

- Integration with Google Calendar for easy event addition.
- Donation forms for both general and event-specific contributions.
- Prompt and accurate notifications for events, volunteer opportunities, and other important updates.

User 3:

- Name: Reshma
- Age: 25
- Occupation: Developer
- Location: Singapore

Pain Points:

- Difficulty in tracking and managing donations effectively.

Preferred Features:

- Volunteer Management: Easily browse and sign up for volunteer opportunities.
- Event Management: Create, manage, and view community events.
- Donation Tracking: Separate donations.
- Adoption System: View and manage pet adoption requests.

(930 words)

Conclusion:

The dashboard page of our non-profit organization's website effectively supports comprehensive event management, providing users with tools to create, edit, delete, and view events. These features significantly enhance our ability to organize and showcase upcoming activities. Here's a summary of our current strengths and areas for improvement:

Strengths:

- Event Management: Users can efficiently create new events with essential details and media uploads, ensuring accurate and comprehensive event information. The editing feature allows for seamless updates, and the deletion functionality helps keep the event list current and relevant.
- User-Friendly Display: Events are presented in an organized card format, making it easy for users to navigate and access important details.
- Volunteer Registration: The system captures not only basic user information but also additional volunteer-specific details, providing a holistic view of each volunteer.
- Donation Flexibility: Users can choose to make donations either for specific events or as general contributions, providing flexibility in how they support the organization.
- Calendar Integration: The ability to add events to Google Calendar enhances user convenience and encourages participation.

Areas for Improvement:

- UI/UX Design: The current interface could be made more engaging to address feedback about the site feeling a bit boring. Enhanced visual design and interactive elements could make the experience more appealing.
- Feature Enhancement: Additional features like detailed event analytics or integrated feedback mechanisms could provide deeper insights into event performance and user satisfaction.
- Performance Optimization: Ensuring fast load times and smooth performance will improve user experience and keep users engaged.
- Accessibility: Implementing accessibility features to support users with disabilities will make the site more inclusive.
- Content Expansion: Completing and integrating the recycling page, blog page, and chat feature will enrich the content and functionality of the site, addressing additional user needs.

If Given the Opportunity to Redo the Project:

- User Personalization: Implement personalized dashboards for users to track their interactions, donations, and volunteer activities more effectively.
- Advanced Analytics: Add analytics tools to track event performance, user engagement, and donation trends to gain deeper insights.
- Interactive Features: Introduce features such as real-time chat support, interactive event maps, and customizable notifications to enhance user engagement.

- Mobile Optimization: Ensure the platform is fully optimized for mobile devices, providing a seamless experience across all screen sizes.
- Enhanced Security: Implement additional security measures, such as multi-factor authentication, to protect user data and transactions.

As we conclude this project, our non-profit organization's website has been developed with a range of functionalities designed to enhance engagement, manage events, and streamline user interactions. I believe this project has laid a solid foundation for our website, and with ongoing improvements and deployment, it will serve as a valuable tool for our organization and its stakeholders.

(440 words)

References:

[i] Clean & Green Singapore. Accessed June 12, 2024. https://www.cgs.gov.sg/.

[ii] "Volunteer as an Individual." Mandai Wildlife Reserve. Accessed June 12, 2024. https://www.mandai.com/en/get-involved/volunteer/individual-volunteer.html.

[iii] National Parks Board. Accessed June 12, 2024. https://www.nparks.gov.sg/.

[iv] "Compassion for All Creatures: SPCA Singapore's Commitment to Animal Welfare." Society for the Prevention of Cruelty to Animals. Accessed June 12, 2024. https://spca.org.sg/.

[v] "Django Python Tutorial: What It Is and Framework - Javatpoint." www.javatpoint.com. Accessed June 12, 2024. https://www.javatpoint.com/django-tutorial.

[vi] "Django." Django Project. Accessed March 10, 2024. https://docs.djangoproject.com/en/5.0/topics/db/queries/#complex-lookups-with-qobjects.

[vii] "Django." Django Project. Accessed March 10, 2024. https://docs.djangoproject.com/en/5.0/ref/templates/language/.

[viii] Mark Otto, Jacob Thornton. "Title of a Longer Featured Blog Post." Blog Template · Bootstrap v5.3. Accessed June 12, 2024. https://getbootstrap.com/docs/5.3/examples/blog/.

[ix] "{% Crispy %} Tag with FormsÁ." django. Accessed March 10, 2024. https://djangocrispy-forms.readthedocs.io/en/latest/crispy_tag_forms.html.

[x] "Django." Django Project. Accessed March 10, 2024. https://docs.djangoproject.com/en/5.0/ref/templates/builtins/.

[xi] "Django." Django Project. Accessed March 10, 2024. https://docs.djangoproject.com/en/5.0/topics/auth/default/.

[xii] Django, Learn. "Django Login, Logout, Signup, Password Change, and Password Reset." Home. Accessed March 10, 2024. https://learndjango.com/tutorials/django-login-andlogout-tutorial.

[xiii] "How Do I Generate 'add to Calendar' Link from Our Own Website?" Google Calendar Community. Accessed July 21, 2024. https://support.google.com/calendar/thread/81344786/how-do-i-generate-add-to-calendar-link-from-our-own-website?hl=en.

[xiv] Puppy for sale - pets plus stafford. Accessed September 9, 2024. https://www.petsplusinc.com/pets-for-sale/puppies.

[xv] "Super Simple Wysiwyg Editor." Summernote. Accessed September 9, 2024. https://summernote.org/getting-started/#google_vignette.

[xvi] Mark Otto, Jacob Thornton. "Cards." · Bootstrap v5.3. Accessed March 10, 2024. https://getbootstrap.com/docs/5.3/components/card/#about.

[xvii] Christie, Tom. "Tutorial 3: Class-Based Views." 3 - Class based views - Django REST framework. Accessed January 7, 2024. https://www.django-restframework.

org/tutorial/3-class-based-views/.

[xviii] Christie, Tom. “Testing.” Testing - Django REST framework. Accessed March 10, 2024. https://www.django-rest-framework.org/api-guide/testing/.

[xix] Real Python. “Testing in Django (Part 1) – Best Practices and Examples.” Real Python, October 6, 2021. https://realpython.com/testing-in-django-part-1-best-practices-and-examples/.

[xx] “Understanding Unit Testing in Python.” BrowserStack, August 8, 2024. https://www.browserstack.com/guide/unit-testing-python.

[xxi] “Testing Tools: Django Documentation.” Django Project. Accessed September 9, 2024. https://docs.djangoproject.com/en/5.1/topics/testing/tools/#django.test.TestCase.

Mark Otto, Jacob Thornton. “Modal.” · Bootstrap v5.3. Accessed September 9, 2024. https://getbootstrap.com/docs/5.3/components/modal/#live-demo.

[xxii] “Sending Email: Django Documentation.” Django Project. Accessed September 9, 2024. https://docs.djangoproject.com/en/5.0/topics/email/#smtp-backend.