Wildlife Conservation Management System
DBS Mini Project

Team Members

CSE C Lab Batch 1

Aditya Baranwal	Roll No. 25	230905178
Anushka Prabhutendolkar	Roll No. 10	230905050

Abstract

This project focuses on building a Wildlife Conservation Management Portal, allowing for real-time data entry and analysis of species, habitats, communities, and threats. The system supports tracking user contributions and enables community-based conservation strategies.
Modifications: The project integrates Flask-based routes for RESTful interaction, uses MySQL as the backend, and includes SQL triggers for automation.

Problem Statement

The problem statement is to create a management system for animal species in India as to where they naturally live, were spotted or are on the brink of extinction. The platform serves the following:
Data Requirements:

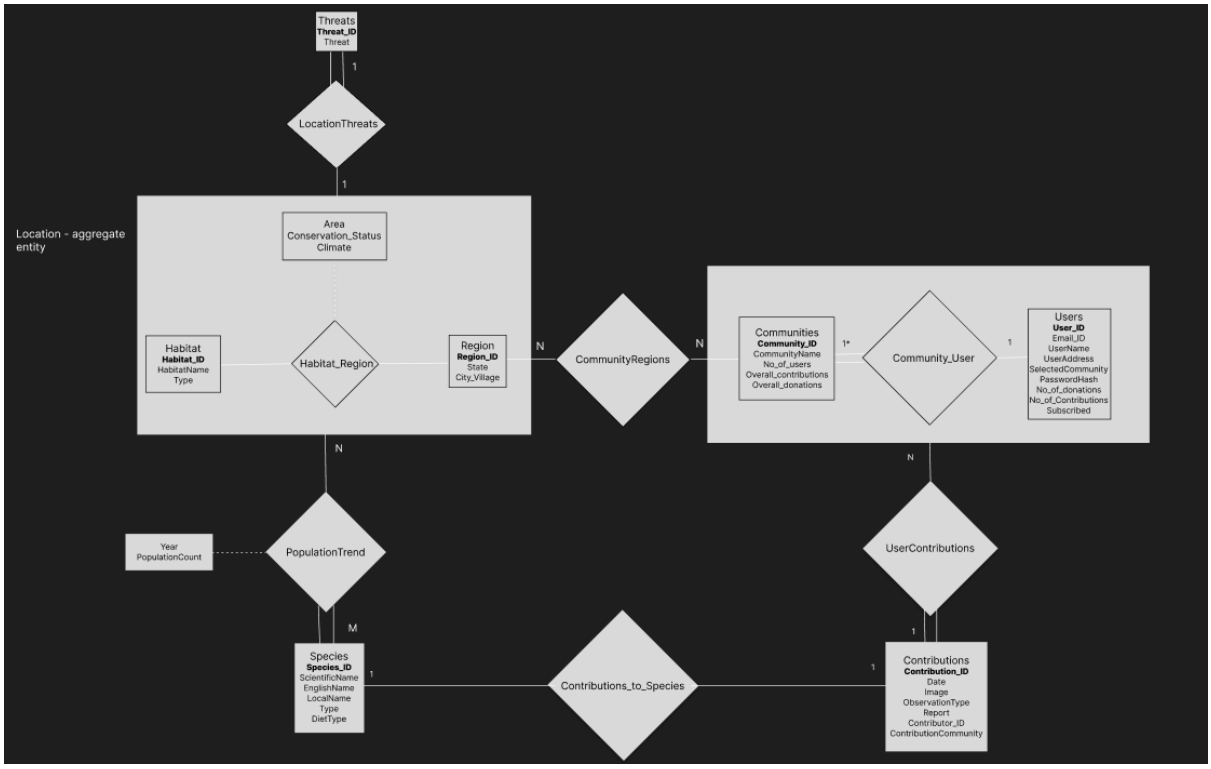
1. Maintain records of species (scientific, local names, type, diet)
2. Track habitats, regions, and mapping of habitats to regions
3. Record contributions from users
4. Auto-increment IDs (PKs of all entities) so as to get uniquely identifiable records
5. Link users to communities and map them to regions
6. Store threats affecting species or locations.

Functional Requirements:

1. Users can log in, contribute sightings, be a part of communities working for this cause
2. Admin can add species and manage threats
3. Community statistics auto-update on contribution
4. Image uploads for sightings
5. SQL triggers for automated updates (e.g., community updates on contribution)

ER Diagram and Relational Tables

ER Diagram



Relational Tables

Entities

Region				
Column	**Datatype**	**Space**	**Primary/Foreign Key**	**Autoincrement**
Region_ID	int		PK	yes
State	varchar	100		
City_Village	varchar	100		

Threats				
Column	**Datatype**	**Space**	**Primary/Foreign Key**	**Autoincrement**
Threat_ID	int		PK	yes
ThreatName	varchar	255		

Location

Column	Datatype	Space	Primary/Foreign Key	Autoincrement
Location_ID	int		PK	yes
Habitat_ID	int		FK	
Region_ID	int		FK	
Area	float			
Climate	varchar	100		

Species

Column	Datatype	Space	Primary/Foreign Key	Autoincrement
Species_ID	int		PK	yes
ScientificName	varchar	255		
EnglishName	varchar	255		
LocalName	varchar	255		
Type	varchar	50		
DietType	varchar	50		

Users

Column	Datatype	Space	Primary/Foreign Key	Autoincrement
User_ID	int		PK	yes
Email_ID	varchar	255		
UserName	varchar	100		
UserAddress	text			
PasswordHash	varchar	255		
No_of_contributions	int			

Communities

Column	Datatype	Space	Primary/Foreign Key	Autoincrement
Community_ID	int		PK	yes
CommunityName	varchar	255		
No_of_Users	int			
Total_contributions	int			

Contributions

Column	Datatype	Space	Primary/Foreign Key	Autoincrement
Contribution_ID	int		PK	yes
Date	date			
Image	blob			
ObservationType	varchar	255		
Report	text			

Relationships

Habitat_Region				
Column	Datatype	Space	Primary/Foreign Key	
Region_ID	int		FK	PK
Habitat_ID	int		FK	
Area	float			
Conservation_Status	varchar	255		
Climate	varchar	100		

LocationThreats				
Column	Datatype	Space	Primary/Foreign Key	
Location_ID	int		FK	PK
Threat_ID	int		FK	

PopulationTrend				
Column	Datatype	Space	Primary/Foreign Key	
Species_ID	int		FK	PK
Location_ID	int		FK	
Year	year			
PopulationCount	int			

UserContributions				
Column	Dataype	Space	Primary/Foreign Key	
Contribution_ID	int		FK	PK
CU_ID	int		FK	

Contributions_to_Species				
Column	Dataype	Space	Primary/Foreign Key	
Contribution_ID	int		FK	PK
Species_ID	int		FK	

DDL Commands to create tables

```
-- Create Habitat Table
CREATE TABLE Habitat (
    Habitat_ID INT PRIMARY KEY AUTO_INCREMENT,
    HabitatName VARCHAR(255) NOT NULL,
    Type VARCHAR(50) NOT NULL
);

-- Create Region Table
CREATE TABLE Region (
    Region_ID INT PRIMARY KEY AUTO_INCREMENT,
    State VARCHAR(100) NOT NULL,
    City_Village VARCHAR(100) NOT NULL
);

-- Create Habitat_Region Relationship Table
CREATE TABLE Habitat_Region (
    Habitat_ID INT,
    Region_ID INT,
    Area FLOAT NOT NULL,
    Conservation_Status VARCHAR(255),
    Climate VARCHAR(100),
    PRIMARY KEY (Habitat_ID, Region_ID),
    FOREIGN KEY (Habitat_ID) REFERENCES Habitat(Habitat_ID),
    FOREIGN KEY (Region_ID) REFERENCES Region(Region_ID)
);

-- Create Threats Table
CREATE TABLE Threats (
    Threat_ID INT PRIMARY KEY AUTO_INCREMENT,
    ThreatName VARCHAR(255) NOT NULL
);

-- Create Location Aggregate Table (HabitatRegion)
CREATE TABLE Location (
    Location_ID INT PRIMARY KEY AUTO_INCREMENT,
    Habitat_ID INT NOT NULL,
    Region_ID INT NOT NULL,
    Area FLOAT NOT NULL,
    Conservation_Status VARCHAR(255),
    Climate VARCHAR(100),
    FOREIGN KEY (Habitat_ID) REFERENCES Habitat(Habitat_ID),
    FOREIGN KEY (Region_ID) REFERENCES Region(Region_ID)
);

-- Create LocationThreats Relationship Table
CREATE TABLE LocationThreats (
    Location_ID INT,
    Threat_ID INT,
    PRIMARY KEY (Location_ID, Threat_ID),
    FOREIGN KEY (Location_ID) REFERENCES Location(Location_ID),
    FOREIGN KEY (Threat_ID) REFERENCES Threats(Threat_ID)
```

```sql
);

-- Create Species Table
CREATE TABLE Species (
    Species_ID INT PRIMARY KEY AUTO_INCREMENT,
    ScientificName VARCHAR(255) NOT NULL,
    EnglishName VARCHAR(255),
    LocalName VARCHAR(255),
    Type VARCHAR(50) NOT NULL,
    DietType VARCHAR(50) NOT NULL
);

-- Create PopulationTrend Relationship Table
CREATE TABLE PopulationTrend (
    Species_ID INT,
    Location_ID INT,
    Year YEAR NOT NULL,
    PopulationCount INT NOT NULL,
    PRIMARY KEY (Species_ID, Location_ID, Year),
    FOREIGN KEY (Species_ID) REFERENCES Species(Species_ID),
    FOREIGN KEY (Location_ID) REFERENCES Location(Location_ID)
);

-- Create Users Table
CREATE TABLE Users (
    User_ID INT PRIMARY KEY AUTO_INCREMENT,
    Email_ID VARCHAR(255) UNIQUE NOT NULL,
    UserName VARCHAR(100) NOT NULL,
    UserAddress TEXT,
    PasswordHash VARCHAR(255) NOT NULL,
    No_of_contributions INT DEFAULT 0,
);

-- Create Communities Table
CREATE TABLE Communities (
    Community_ID INT PRIMARY KEY AUTO_INCREMENT,
    CommunityName VARCHAR(255) NOT NULL,
    No_of_Users INT DEFAULT 0,
    Total_contributions INT DEFAULT 0
);

-- Create Community_User Relationship Table
CREATE TABLE Community_User (
    User_ID INT,
    Community_ID INT,
    PRIMARY KEY (User_ID, Community_ID),
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (Community_ID) REFERENCES Communities(Community_ID)
);

-- Create CommunityRegions Relationship Table
CREATE TABLE CommunityRegions (
    Region_ID INT,
```

```
    Community_ID INT,
    PRIMARY KEY (Region_ID, Community_ID),
    FOREIGN KEY (Region_ID) REFERENCES Region(Region_ID),
    FOREIGN KEY (Community_ID) REFERENCES Communities(Community_ID)
);

-- Create Communities_Users Aggregate Table
CREATE TABLE Communities_Users (
    CU_ID INT PRIMARY KEY AUTO_INCREMENT,
    Community_ID INT NOT NULL,
    User_ID INT NOT NULL,
    No_of_contributions INT DEFAULT 0,
    FOREIGN KEY (Community_ID) REFERENCES Communities(Community_ID),
    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);

-- Create Contributions Table
CREATE TABLE Contributions (
    Contribution_ID INT PRIMARY KEY AUTO_INCREMENT,
    Date DATE NOT NULL,
    Image VARCHAR(255),
    ObservationType VARCHAR(255),
    Report TEXT
);

-- Create UserContributions Relationship Table
CREATE TABLE UserContributions (
    Contribution_ID INT,
    CU_ID INT,
    PRIMARY KEY (Contribution_ID, CU_ID),
    FOREIGN KEY (Contribution_ID) REFERENCES
Contributions(Contribution_ID),
    FOREIGN KEY (CU_ID) REFERENCES Communities_Users(CU_ID)
);

-- Create Contributions_to_Species Relationship Table
CREATE TABLE Contributions_to_Species (
    Contribution_ID INT,
    Species_ID INT,
    PRIMARY KEY (Contribution_ID, Species_ID),
    FOREIGN KEY (Contribution_ID) REFERENCES
Contributions(Contribution_ID),
    FOREIGN KEY (Species_ID) REFERENCES Species(Species_ID)
);
```

List of SQL Queries

```
--Inserting into tables
INSERT INTO Species (ScientificName, EnglishName, LocalName, Type,
DietType) VALUES
('Panthera tigris tigris', 'Bengal Tiger', 'Baagh', 'Mammal', 'Carnivore'),
('Elephas maximus indicus', 'Indian Elephant', 'Hathi', 'Mammal',
'Herbivore'),
('Python molurus', 'Indian Rock Python', 'Ajgar', 'Reptile', 'Carnivore'),
('Pavo cristatus', 'Indian Peafowl', 'Mor', 'Bird', 'Omnivore'),
('Gavialis gangeticus', 'Gharial', 'Gharial', 'Reptile', 'Carnivore'),
('Bos gaurus', 'Indian Bison (Gaur)', 'Gaur', 'Mammal', 'Herbivore'),
('Macaca mulatta', 'Rhesus Macaque', 'Bandar', 'Mammal', 'Omnivore'),
('Varanus bengalensis', 'Bengal Monitor', 'Goh', 'Reptile', 'Carnivore'),
('Corvus splendens', 'House Crow', 'Kauwa', 'Bird', 'Omnivore'),
('Axis axis', 'Chital (Spotted Deer)', 'Chital', 'Mammal', 'Herbivore'),
('Heteropneustes fossilis', 'Stinging Catfish', 'Singhi', 'Fish',
'Omnivore'),
('Rhinoceros unicornis', 'Indian Rhinoceros', 'Gainda', 'Mammal',
'Herbivore'),
('Bufo melanostictus', 'Common Indian Toad', 'Bhindi Mendak', 'Amphibian',
'Insectivore'),
('Melursus ursinus', 'Sloth Bear', 'Bhaloo', 'Mammal', 'Omnivore'),
('Accipiter badius', 'Shikra', 'Shikra', 'Bird', 'Carnivore'),
('Duttaphrynus melanostictus', 'Asian Common Toad', 'Mendak', 'Amphibian',
'Insectivore'),
('Psittacula krameri', 'Rose-ringed Parakeet', 'Tota', 'Bird',
'Herbivore'),
('Neofelis nebulosa', 'Clouded Leopard', 'Dhundli Chita', 'Mammal',
'Carnivore'),
('Ichthyophis beddomei', 'Beddome’s Caecilian', 'Jal Saap', 'Amphibian',
'Carnivore'),
('Rattus rattus', 'Black Rat', 'Chooha', 'Mammal', 'Omnivore');


INSERT INTO Habitat (HabitatName, Type) VALUES
('Sundarbans Mangrove Forest', 'Mangrove'),
('Western Ghats', 'Rainforest'),
('Thar Desert', 'Desert'),
('Himalayan Alpine Meadows', 'Alpine'),
('Kaziranga Floodplains', 'Grassland'),
('Rann of Kutch', 'Salt Marsh'),
('Keoladeo National Park Wetlands', 'Wetland'),
('Nilgiri Hills', 'Montane'),
('Coringa Mangrove Reserve', 'Mangrove'),
('Jim Corbett Forest', 'Deciduous'),
('Andaman Tropical Rainforest', 'Rainforest'),
('Chilika Lake', 'Brackish'),
('Valley of Flowers', 'Alpine'),
('Deccan Plateau', 'Dry Forest'),
('Sathyamangalam Forests', 'Dry Deciduous'),
('Loktak Lake', 'Freshwater'),
('Anamalai Tiger Reserve', 'Moist Forest'),
```

```
('Nallamala Forest', 'Tropical'),
('Dachigam National Park', 'Temperate'),
('Great Himalayan National Park', 'Coniferous');

INSERT INTO Region (State, City_Village) VALUES
('West Bengal', 'Sundarbans'),
('Assam', 'Kaziranga'),
('Rajasthan', 'Jaisalmer'),
('Uttarakhand', 'Nainital'),
('Kerala', 'Wayanad'),
('Tamil Nadu', 'Pollachi'),
('Gujarat', 'Bhuj'),
('Andhra Pradesh', 'Coringa'),
('Karnataka', 'Bandipur'),
('Odisha', 'Chilika'),
('Himachal Pradesh', 'Kullu'),
('Meghalaya', 'Cherrapunji'),
('Madhya Pradesh', 'Kanha'),
('Maharashtra', 'Tadoba'),
('Manipur', 'Moirang'),
('Arunachal Pradesh', 'Ziro'),
('Sikkim', 'Yuksom'),
('Jammu & Kashmir', 'Dachigam'),
('Bihar', 'Valmikinagar'),
('Punjab', 'Harike');

INSERT INTO Communities (CommunityName, No_of_Users, Overall_contributions,
Overall_donations) VALUES
('Wildlife Conservation India', 1500, 120, 20000.50),
('Green Earth Foundation', 1200, 250, 35000.75),
('Clean Rivers Initiative', 800, 180, 15000.20),
('Save the Tigers', 2200, 350, 50000.00),
('Forest Conservation Society', 500, 90, 7000.10),
('Earth Warriors', 3500, 500, 100000.00),
('Plastic-Free India', 1500, 450, 32000.00),
('Ganga Rejuvenation Project', 2000, 300, 15000.75),
('Urban Green Spaces India', 600, 130, 8000.50),
('Eco Warriors Network', 1000, 200, 25000.00),
('Himalayan Wildlife Rescue', 400, 75, 5000.30),
('Clean Air India', 1100, 160, 12000.00),
('Coastal Conservation Trust', 700, 120, 15000.80),
('Earth First India', 3000, 600, 80000.00),
('Green India Foundation', 1300, 220, 15000.90),
('Save Our Soil Initiative', 800, 100, 9000.00),
('Water Conservation Council', 1000, 250, 22000.00),
('Clean India Movement', 2500, 400, 65000.50),
('Sustainable Agriculture India', 600, 150, 11000.00),
('Nature Conservation Collective', 2000, 350, 45000.00);



--Insert data for population trend
```

```
INSERT INTO PopulationTrend (Species_ID, Location_ID, Year,
PopulationCount) VALUES
(1, 1, 2020, 647),
(1, 1, 2021, 656),
(1, 1, 2022, 646),
(1, 1, 2023, 659),
(1, 1, 2024, 674),
(1, 4, 2020, 542),
(1, 4, 2021, 534),
(1, 4, 2022, 549),
(1, 4, 2023, 564),
(1, 4, 2024, 562),
(2, 2, 2020, 157),
(2, 2, 2021, 172),
(2, 2, 2022, 163),
(2, 2, 2023, 153),
(2, 2, 2024, 144),
(2, 1, 2020, 313),
(2, 1, 2021, 331),
(2, 1, 2022, 341),
(2, 1, 2023, 358),
(2, 1, 2024, 369),
(3, 5, 2020, 529),
(3, 5, 2021, 543),
(3, 5, 2022, 558),
(3, 5, 2023, 555),
(3, 5, 2024, 555),
(3, 2, 2020, 603),
(3, 2, 2021, 601),
(3, 2, 2022, 594),
(3, 2, 2023, 591),
(3, 2, 2024, 597),
(4, 1, 2020, 889),
(4, 1, 2021, 893),
(4, 1, 2022, 892),
(4, 1, 2023, 885),
(4, 1, 2024, 880),
(4, 4, 2020, 227),
(4, 4, 2021, 232),
(4, 4, 2022, 231),
(4, 4, 2023, 241),
(4, 4, 2024, 252),
(5, 2, 2020, 146),
(5, 2, 2021, 152),
(5, 2, 2022, 162),
(5, 2, 2023, 167),
(5, 2, 2024, 174),
(5, 3, 2020, 932),
(5, 3, 2021, 928),
(5, 3, 2022, 934),
(5, 3, 2023, 929),
(5, 3, 2024, 928),
(6, 5, 2020, 953),
```

```
(6, 5, 2021, 950),
(6, 5, 2022, 955),
(6, 5, 2023, 945),
(6, 5, 2024, 961),
(6, 2, 2020, 841),
(6, 2, 2021, 841),
(6, 2, 2022, 833),
(6, 2, 2023, 823),
(6, 2, 2024, 833),
(7, 4, 2020, 812),
(7, 4, 2021, 831),
(7, 4, 2022, 831),
(7, 4, 2023, 827),
(7, 4, 2024, 819),
(7, 3, 2020, 384),
(7, 3, 2021, 390),
(7, 3, 2022, 394),
(7, 3, 2023, 401),
(7, 3, 2024, 413),
(8, 5, 2020, 78),
(8, 5, 2021, 77),
(8, 5, 2022, 78),
(8, 5, 2023, 92),
(8, 5, 2024, 91),
(8, 2, 2020, 416),
(8, 2, 2021, 430),
(8, 2, 2022, 437),
(8, 2, 2023, 449),
(8, 2, 2024, 460),
(9, 1, 2020, 764),
(9, 1, 2021, 775),
(9, 1, 2022, 783),
(9, 1, 2023, 789),
(9, 1, 2024, 796),
(9, 4, 2020, 160),
(9, 4, 2021, 164),
(9, 4, 2022, 159),
(9, 4, 2023, 154),
(9, 4, 2024, 169),
(10, 3, 2020, 361),
(10, 3, 2021, 371),
(10, 3, 2022, 363),
(10, 3, 2023, 361),
(10, 3, 2024, 370),
(10, 5, 2020, 317),
(10, 5, 2021, 329),
(10, 5, 2022, 332),
(10, 5, 2023, 341),
(10, 5, 2024, 337),
(11, 4, 2020, 742),
(11, 4, 2021, 744),
(11, 4, 2022, 741),
(11, 4, 2023, 734),
```

```
(11, 4, 2024, 738),
(11, 3, 2020, 569),
(11, 3, 2021, 589),
(11, 3, 2022, 595),
(11, 3, 2023, 612),
(11, 3, 2024, 614),
(12, 2, 2020, 985),
(12, 2, 2021, 983),
(12, 2, 2022, 979),
(12, 2, 2023, 974),
(12, 2, 2024, 987),
(12, 1, 2020, 891),
(12, 1, 2021, 895),
(12, 1, 2022, 912),
(12, 1, 2023, 920),
(12, 1, 2024, 917),
(13, 5, 2020, 986),
(13, 5, 2021, 1001),
(13, 5, 2022, 994),
(13, 5, 2023, 989),
(13, 5, 2024, 985),
(13, 3, 2020, 185),
(13, 3, 2021, 190),
(13, 3, 2022, 181),
(13, 3, 2023, 178),
(13, 3, 2024, 172),
(14, 4, 2020, 719),
(14, 4, 2021, 715),
(14, 4, 2022, 735),
(14, 4, 2023, 743),
(14, 4, 2024, 733),
(14, 5, 2020, 872),
(14, 5, 2021, 882),
(14, 5, 2022, 885),
(14, 5, 2023, 876),
(14, 5, 2024, 890),
(15, 1, 2020, 857),
(15, 1, 2021, 864),
(15, 1, 2022, 876),
(15, 1, 2023, 887),
(15, 1, 2024, 895),
(15, 5, 2020, 666),
(15, 5, 2021, 671),
(15, 5, 2022, 668),
(15, 5, 2023, 678),
(15, 5, 2024, 687),
(16, 2, 2020, 526),
(16, 2, 2021, 518),
(16, 2, 2022, 532),
(16, 2, 2023, 545),
(16, 2, 2024, 549),
(16, 3, 2020, 83),
(16, 3, 2021, 100),
```

```
(16, 3, 2022, 98),
(16, 3, 2023, 114),
(16, 3, 2024, 109),
(17, 5, 2020, 594),
(17, 5, 2021, 599),
(17, 5, 2022, 617),
(17, 5, 2023, 608),
(17, 5, 2024, 626),
(17, 2, 2020, 253),
(17, 2, 2021, 252),
(17, 2, 2022, 267),
(17, 2, 2023, 273),
(17, 2, 2024, 290),
(18, 1, 2020, 192),
(18, 1, 2021, 189),
(18, 1, 2022, 188),
(18, 1, 2023, 194),
(18, 1, 2024, 212),
(18, 4, 2020, 244),
(18, 4, 2021, 261),
(18, 4, 2022, 261),
(18, 4, 2023, 271),
(18, 4, 2024, 289),
(19, 2, 2020, 254),
(19, 2, 2021, 253),
(19, 2, 2022, 243),
(19, 2, 2023, 246),
(19, 2, 2024, 243),
(19, 4, 2020, 322),
(19, 4, 2021, 334),
(19, 4, 2022, 334),
(19, 4, 2023, 327),
(19, 4, 2024, 317),
(20, 2, 2020, 881),
(20, 2, 2021, 874),
(20, 2, 2022, 886),
(20, 2, 2023, 882),
(20, 2, 2024, 899),
(20, 5, 2020, 494),
(20, 5, 2021, 505),
(20, 5, 2022, 504),
(20, 5, 2023, 510),
(20, 5, 2024, 510);
```

PL/SQL

```
--Trigger that fires after a new row is inserted into the Users table.

--It checks the new field SelectedCommunity and, if it is not "None",
inserts a record into the Community_User table

DELIMITER $$

CREATE TRIGGER after_insert_users_community

AFTER INSERT ON Users

FOR EACH ROW

BEGIN

    -- Check if the user selected a community (i.e. the value is not
'None')

    IF NEW.SelectedCommunity IS NOT NULL AND NEW.SelectedCommunity <>
'None' THEN

        INSERT INTO Community_User (User_ID, Community_ID)

        VALUES (NEW.User_ID, NEW.SelectedCommunity);

    END IF;

END$$

DELIMITER ;


--Trigger to increment no of users in communities table after inserting a
community user in the database

DELIMITER $$

CREATE TRIGGER after_insert_community_user

    AFTER INSERT ON Community_User

    FOR EACH ROW

    BEGIN

        UPDATE Communities

        SET No_of_Users = No_of_Users + 1

        WHERE Community_ID = NEW.Community_ID;

    END$$

DELIMITER ;


-- AFTER INSERT trigger on the Contributions table. The trigger will:

-- Retrieve the stored community of the contributor from the Users table.
```

```
-- Always update the user's No_of_contributions by +1.

-- If the ContributionCommunity (from the form) is not "None" and matches
the user's stored SelectedCommunity,

-- then also update the corresponding community’s Total_contributions by
+1.


DELIMITER $$


CREATE TRIGGER after_insert_contribution

AFTER INSERT ON Contributions

FOR EACH ROW

BEGIN

    DECLARE userCommunity VARCHAR(10);


    -- Get the user’s stored community from the Users table.

    SELECT SelectedCommunity INTO userCommunity

      FROM Users

     WHERE User_ID = NEW.Contributor_ID;


    -- Always increment the user’s contribution count.

    UPDATE Users

       SET No_of_contributions = No_of_contributions + 1

     WHERE User_ID = NEW.Contributor_ID;


    -- If the chosen community is not 'None' and matches the user's stored
community,

    -- update the community’s total contributions.

    IF NEW.ContributionCommunity <> 'None' AND NEW.ContributionCommunity =
userCommunity THEN

        UPDATE Communities

           SET Total_contributions = Total_contributions + 1

         WHERE Community_ID = NEW.ContributionCommunity;

    END IF;

END$$
```

```
DELIMITER ;


--AFTER DELETE Trigger on the Community_User table so that when a user-
community link is removed the community's user count is updated

DELIMITER $$


CREATE TRIGGER after_delete_community_user

AFTER DELETE ON Community_User

FOR EACH ROW

BEGIN

    UPDATE Communities

    SET No_of_Users = No_of_Users - 1

    WHERE Community_ID = OLD.Community_ID;

END$$


DELIMITER ;
```

Python Code for Functional Design

The different modules for the same are:

config.py:

```
import os

basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'mysql://adiand:ludo.1234@localhost/wildlife_in_a_flask'
    SQLALCHEMY_TRACK_MODIFICATIONS = False

    # New upload configuration
    UPLOAD_FOLDER = os.path.join(basedir, 'static', 'uploads')
    ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
```

extensions.py:

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager

db = SQLAlchemy()
login = LoginManager()
```

app.py

```
# First, apply the monkey patch.
import patch_werkzeug  # ensure this is in the project root and imported
first

from flask import Flask
from config import Config
from extensions import db, login
# We are leaving out flask-migrate for now
# from flask_migrate import Migrate

def create_app(config_class=Config):
    app = Flask(__name__)
    app.config.from_object(config_class)

    db.init_app(app)
    login.init_app(app)

    # Register blueprints
    from routes import main_routes, user_routes, species_routes,
contribution_routes
    app.register_blueprint(main_routes.bp)
    app.register_blueprint(user_routes.bp)
    app.register_blueprint(species_routes.bp)
```

```
    app.register_blueprint(contribution_routes.bp)

    return app

app = create_app()

# Define the user loader for Flask-Login
from models import Users  # Import here to avoid circular dependencies
@login.user_loader
def load_user(user_id):
    return Users.query.get(int(user_id))

if __name__ == '__main__':
    app.run(debug=True)
```

models.py

```
from extensions import db
from flask_login import UserMixin

class Habitat(db.Model):
    __tablename__ = 'Habitat'
    Habitat_ID = db.Column(db.Integer, primary_key=True)
    HabitatName = db.Column(db.String(255), nullable=False)
    Type = db.Column(db.String(50), nullable=False)

class Region(db.Model):
    __tablename__ = 'Region'
    Region_ID = db.Column(db.Integer, primary_key=True)
    State = db.Column(db.String(100), nullable=False)
    City_Village = db.Column(db.String(100), nullable=False)

class HabitatRegion(db.Model):
    __tablename__ = 'Location'
    Location_ID = db.Column(db.Integer, primary_key=True)
    Habitat_ID = db.Column(db.Integer, db.ForeignKey('habitat.Habitat_ID'),
nullable=False)
    Region_ID = db.Column(db.Integer, db.ForeignKey('region.Region_ID'),
nullable=False)
    Area = db.Column(db.Float, nullable=False)
    Conservation_Status = db.Column(db.String(255))
    Climate = db.Column(db.String(100))

class Threats(db.Model):
    __tablename__ = 'Threats'
    Threat_ID = db.Column(db.Integer, primary_key=True)
    ThreatName = db.Column(db.String(255), nullable=False)

class LocationThreats(db.Model):
    __tablename__ = 'LocationThreats'
    Location_ID = db.Column(db.Integer,
db.ForeignKey('location.Location_ID'), primary_key=True)
```

```
    Threat_ID = db.Column(db.Integer, db.ForeignKey('threats.Threat_ID'),
primary_key=True)

class Species(db.Model):
    __tablename__ = 'Species'
    Species_ID = db.Column(db.Integer, primary_key=True)
    ScientificName = db.Column(db.String(255), nullable=False)
    EnglishName = db.Column(db.String(255))
    LocalName = db.Column(db.String(255))
    Type = db.Column(db.String(50), nullable=False)
    DietType = db.Column(db.String(50), nullable=False)

class PopulationTrend(db.Model):
    __tablename__ = 'PopulationTrend'
    Species_ID = db.Column(db.Integer, db.ForeignKey('species.Species_ID'),
primary_key=True)
    Location_ID = db.Column(db.Integer,
db.ForeignKey('location.Location_ID'), primary_key=True)
    Year = db.Column(db.String(4), primary_key=True)  # stored as string
(e.g., "2025")
    PopulationCount = db.Column(db.Integer, nullable=False)

class Users(UserMixin, db.Model):
    __tablename__ = 'Users'
    User_ID = db.Column(db.Integer, primary_key=True, autoincrement=True)
    Email_ID = db.Column(db.String(255), unique=True, nullable=False)
    UserName = db.Column(db.String(100), nullable=False)
    UserAddress = db.Column(db.Text)
    SelectedCommunity = db.Column(db.String(10))
    PasswordHash = db.Column(db.String(255), nullable=False)
    No_of_donations = db.Column(db.Integer, default=0)
    No_of_contributions = db.Column(db.Integer, default=0)
    Subscribed = db.Column(db.Boolean, default=False)

    def get_id(self):
        return str(self.User_ID)

class Communities(db.Model):
    __tablename__ = 'Communities'
    Community_ID = db.Column(db.Integer, primary_key=True)
    CommunityName = db.Column(db.String(255), nullable=False)
    No_of_Users = db.Column(db.Integer, default=0)
    Total_contributions = db.Column(db.Integer, default=0)
    Total_donations = db.Column(db.Float, default=0.0)

class CommunityUser(db.Model):
    __tablename__ = 'Community_User'
    User_ID = db.Column(db.Integer, db.ForeignKey('users.User_ID'),
primary_key=True)
    Community_ID = db.Column(db.Integer,
db.ForeignKey('communities.Community_ID'), primary_key=True)

class CommunityRegions(db.Model):
```

```
    __tablename__ = 'CommunityRegions'
    Region_ID = db.Column(db.Integer, db.ForeignKey('region.Region_ID'),
primary_key=True)
    Community_ID = db.Column(db.Integer,
db.ForeignKey('communities.Community_ID'), primary_key=True)

class CommunitiesUsers(db.Model):
    __tablename__ = 'Communities_Users'
    CU_ID = db.Column(db.Integer, primary_key=True)
    Community_ID = db.Column(db.Integer,
db.ForeignKey('communities.Community_ID'), nullable=False)
    User_ID = db.Column(db.Integer, db.ForeignKey('users.User_ID'),
nullable=False)
    No_of_contributions = db.Column(db.Integer, default=0)

class Contributions(db.Model):
    __tablename__ = 'Contributions'
    Contribution_ID = db.Column(db.Integer, primary_key=True)
    Date = db.Column(db.Date, nullable=False)
    Image = db.Column(db.String(255))
    ObservationType = db.Column(db.String(255))
    Report = db.Column(db.Text)
    Contributor_ID = db.Column(db.Integer, db.ForeignKey('Users.User_ID'),
nullable=False)
    ContributionCommunity = db.Column(db.String(10))  # will store the
drop-down selected community

class UserContributions(db.Model):
    __tablename__ = 'UserContributions'
    Contribution_ID = db.Column(db.Integer,
db.ForeignKey('contributions.Contribution_ID'), primary_key=True)
    CU_ID = db.Column(db.Integer, db.ForeignKey('communities_users.CU_ID'),
primary_key=True)

class ContributionsToSpecies(db.Model):
    __tablename__ = 'Contributions_to_Species'
    Contribution_ID = db.Column(db.Integer,
db.ForeignKey('contributions.Contribution_ID'), primary_key=True)
    Species_ID = db.Column(db.Integer, db.ForeignKey('species.Species_ID'),
primary_key=True)

class Donations(db.Model):
    __tablename__ = 'Donations'
    Donation_ID = db.Column(db.Integer, primary_key=True)
    Transaction_ID = db.Column(db.String(255))
    MemberType = db.Column(db.String(50))
    DonationAmount = db.Column(db.Float, nullable=False)
    DonationDate = db.Column(db.Date, nullable=False)

class UserDonations(db.Model):
    __tablename__ = 'UserDonations'
    Donation_ID = db.Column(db.Integer,
db.ForeignKey('donations.Donation_ID'), primary_key=True)
```

```
    CU_ID = db.Column(db.Integer, db.ForeignKey('communities_users.CU_ID'),
primary_key=True)

class NewslettersUpdates(db.Model):
    __tablename__ = 'Newsletters_Updates'
    Newsletter_ID = db.Column(db.Integer, primary_key=True)
    Title = db.Column(db.String(255), nullable=False)
    Content = db.Column(db.Text, nullable=False)
    Post_Date = db.Column(db.Date, nullable=False)
    Author = db.Column(db.String(255))
```

forms.py:

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, TextAreaField, SubmitField,
DateField, SelectField
from wtforms.validators import DataRequired, EqualTo, Length
from flask_wtf.file import FileField, FileAllowed

class LoginForm(FlaskForm):
    email = StringField('Email address', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Sign In')

class SignupForm(FlaskForm):
    username = StringField('User Name', validators=[DataRequired(),
Length(1, 64)])
    email = StringField('Email address', validators=[DataRequired()])
    address = TextAreaField('Address')  # remains the same for free-form
address input
    password = PasswordField('Password', validators=[DataRequired(),
Length(min=8)])
    password2 = PasswordField('Repeat Password',
                              validators=[DataRequired(),
EqualTo('password', message="Passwords must match.")])
    community = SelectField('Community', choices=[],
validators=[DataRequired()])  # Dropdown for community selection
    submit = SubmitField('Sign Up')

class ContributionForm(FlaskForm):
    date = DateField('Date', format='%Y-%m-%d',
validators=[DataRequired()])
    observation_type = SelectField(
        'Observation Type',
        choices=[
            ('population_update', 'Population Update'),
            ('threat_report', 'Threat Report'),
            ('general_observation', 'General Observation')
        ],
        validators=[DataRequired()]
    )
    image = FileField('Upload Image', validators=[FileAllowed(['jpg',
'jpeg', 'png', 'gif'], 'Images only!')])
```

```
    report = TextAreaField('Report', validators=[DataRequired()])
    community = SelectField('Your Community', choices=[],
validators=[DataRequired()])
    submit = SubmitField('Submit Contribution')
```

The routes used were:

contributions_routes.py

```
# routes/contribution_routes.py
from flask import Blueprint, render_template, redirect, url_for, flash,
current_app, request
from flask_login import login_required, current_user
from extensions import db
from models import Contributions
from forms import ContributionForm
from werkzeug.utils import secure_filename
import os
from sqlalchemy import text

bp = Blueprint('contribution_routes', __name__, url_prefix='/contribution')

def allowed_file(filename):
    # simple check for extension
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in
current_app.config['ALLOWED_EXTENSIONS']

@bp.route('/new', methods=['GET', 'POST'])
@login_required
def new_contribution():
    form = ContributionForm()

    # Populate the community drop-down list (similar to before).
    community_region_list = db.session.execute(
        text('''
          SELECT c.Community_ID, c.CommunityName, r.State, r.City_Village
          FROM Communities c
          JOIN CommunityRegions cr ON c.Community_ID = cr.Community_ID
          JOIN Region r ON cr.Region_ID = r.Region_ID
        ''')
    ).fetchall()

    dropdown_options = [('None', 'None')]
    for row in community_region_list:
        label = f"{row.CommunityName} - {row.State}, {row.City_Village}"
        dropdown_options.append((str(row.Community_ID), label))
    form.community.choices = dropdown_options

    if form.validate_on_submit():
        # Process the file upload.
        image_path = None
```

```
        if form.image.data:
            file = form.image.data
            # Check if the file has an allowed extension.
            if file and allowed_file(file.filename):
                filename = secure_filename(file.filename)
                # Save the file to the configured upload folder.
                upload_folder = current_app.config['UPLOAD_FOLDER']
                file_path = os.path.join(upload_folder, filename)
                file.save(file_path)

                # Store the relative path to the file (accessible via
url_for('static', ...)).
                image_path = os.path.join('uploads', filename)
            else:
                flash('Invalid image file. Please upload a valid image.')
                return render_template('contribution.html', form=form)

        # Create the contribution record.
        contribution = Contributions(
            Date=form.date.data,
            Image=image_path,
            ObservationType=form.observation_type.data,
            Report=form.report.data,
            Contributor_ID=current_user.User_ID,
            ContributionCommunity=form.community.data  # either a valid
community id or 'None'
        )
        db.session.add(contribution)
        db.session.commit()
        flash('Contribution added successfully!')
        return redirect(url_for('user_routes.dashboard'))
    return render_template('contribution.html', form=form)
```

main_routes.py

```
from flask import Blueprint, render_template

bp = Blueprint('main_routes', __name__)

@bp.route('/')
def home():
    return render_template('home.html')
```

species_routes.py

```
from flask import Blueprint, render_template, request
from models import Species
bp = Blueprint('species_routes', __name__, url_prefix='/species')

@bp.route('/')
def species():
    # For now, fetch all species.
    species_list = Species.query.all()
```

```
    return render_template('species.html', species=species_list)
```

user_routes.py

```
from flask import Blueprint, render_template, redirect, url_for, flash,
request
from extensions import db
from models import Contributions, Users
from werkzeug.security import generate_password_hash, check_password_hash
from flask_login import login_user, logout_user, login_required,
current_user
from forms import LoginForm, SignupForm
from sqlalchemy import text

bp = Blueprint('user_routes', __name__, url_prefix='/user')

@bp.route('/signin', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = Users.query.filter_by(Email_ID=form.email.data).first()
        if user and check_password_hash(user.PasswordHash,
form.password.data):
            login_user(user)
            return redirect(url_for('main_routes.home'))
        else:
            flash('Invalid email or password')
    return render_template('signin.html', form=form)

@bp.route('/signup', methods=['GET', 'POST'])
def signup():
    # Query to get community info from Communities and Region (via
CommunityRegions join)
    community_region_list = db.session.execute(
    text('''
    SELECT c.Community_ID, c.CommunityName, r.State, r.City_Village
    FROM Communities c
    JOIN CommunityRegions cr ON c.Community_ID = cr.Community_ID
    JOIN Region r ON cr.Region_ID = r.Region_ID
    ''')
    ).fetchall()

    # Build the dropdown options: add an option for "None" first.
    dropdown_options = [('None', 'None')]
    for row in community_region_list:
        label = f"{row.CommunityName} - {row.State}, {row.City_Village}"
        dropdown_options.append((str(row.Community_ID), label))  # convert
ID to string for consistency

    form = SignupForm()
    # Set the choices dynamically
    form.community.choices = dropdown_options
```

```
    if form.validate_on_submit():
        hashed_password = generate_password_hash(form.password.data)
        # Create a new user; note we store:
        # - The free-form address into UserAddress
        # - The selected community dropdown value into SelectedCommunity.
        new_user = Users(
            Email_ID=form.email.data,
            UserName=form.username.data,
            UserAddress=form.address.data,
            SelectedCommunity=form.community.data,  # contains the
community ID or 'None'
            PasswordHash=hashed_password
        )
        db.session.add(new_user)
        db.session.commit()
        flash('Account created successfully.')
        return redirect(url_for('user_routes.login'))
    return render_template('signup.html', form=form)

@bp.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('main_routes.home'))

@bp.route('/dashboard')
@login_required
def dashboard():
    return render_template('dashboard.html')

@bp.route('/my_contributions', methods=['GET'])
@login_required
def view_contributions():
    # Query contributions made by the current logged in user.
    contributions =
Contributions.query.filter_by(Contributor_ID=current_user.User_ID).all()
    return render_template('dashboard_contributions.html',
contributions=contributions)

@bp.route('/delete_account', methods=['GET', 'POST'])
@login_required
def delete_account():
    if request.method == 'POST':
        # Delete the current user. Because foreign keys are set with ON
DELETE CASCADE,
        # associated records in Community_User will be removed
automatically.
        db.session.delete(current_user)
        db.session.commit()
        flash('Your account has been deleted.', 'info')
        logout_user()
        return redirect(url_for('main_routes.home'))
    # GET request: Render a confirmation page.
```

```
    return render_template('delete_account.html')


For security features like password hashing:
patch_werkzeug.py
# patch_werkzeug.py
import werkzeug.urls
import urllib.parse

if not hasattr(werkzeug.urls, 'url_quote'):
    werkzeug.urls.url_quote = urllib.parse.quote

if not hasattr(werkzeug.urls, 'url_decode'):
    werkzeug.urls.url_decode = urllib.parse.unquote_plus

if not hasattr(werkzeug.urls, 'url_encode'):
    werkzeug.urls.url_encode = urllib.parse.urlencode
```

UI Design

Code for UI design:

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>{% block title %}Wildlife Conservation{% endblock %}</title>
  <!-- Bootstrap CSS -->
  <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css
" rel="stylesheet">
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
  {% block head %}{% endblock %}
</head>
<body>
  <!-- Navigation Bar -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="{{ url_for('main_routes.home')
}}">Wildlife Conservation</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item"><a class="nav-link" href="{{
url_for('species_routes.species') }}">Species</a></li>
        <li class="nav-item"><a class="nav-link" href="{{
url_for('user_routes.dashboard') }}">Dashboard</a></li>
        {% if current_user.is_authenticated %}
          <li class="nav-item"><a class="nav-link" href="{{
url_for('user_routes.logout') }}">Logout</a></li>
          <li class="nav-item">
            <a class="nav-link" href="{{
url_for('user_routes.delete_account') }}"
               onclick="return confirm('Are you sure you want to delete
your account? This action cannot be undone.');">
              Delete Account
            </a>
          </li>
        {% else %}
          <li class="nav-item"><a class="nav-link" href="{{
url_for('user_routes.login') }}">Sign In</a></li>
          <li class="nav-item"><a class="nav-link" href="{{
url_for('user_routes.signup') }}">Sign Up</a></li>
        {% endif %}
      </ul>
```

```
    </div>
  </nav>

  <div class="container mt-4">
    {% with messages = get_flashed_messages() %}
      {% if messages %}
        <div class="alert alert-info">
          {% for message in messages %}
            {{ message }}<br>
          {% endfor %}
        </div>
      {% endif %}
    {% endwith %}
    {% block content %}{% endblock %}
  </div>

  <!-- Bootstrap JS, Popper.js, and jQuery -->
  <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js">
</script>
  <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"><
/script>
  {% block scripts %}{% endblock %}
</body>
</html>
```

Dashboard

dashboard.html

```
{% extends "base.html" %}
{% block title %}Dashboard{% endblock %}
{% block content %}
<h2>Dashboard</h2>
<p>Welcome, {{ current_user.UserName }}!</p>
<div class="list-group">
  <a href="{{ url_for('user_routes.view_contributions') }}" class="list-
group-item list-group-item-action">View Contributions</a>
  <a href="{{ url_for('contribution_routes.new_contribution') }}"
class="list-group-item list-group-item-action">Make a New
Contribution/Report</a>
</div>
{% endblock %}
```

dashboard_contributions.html

```
{% extends "base.html" %}
{% block title %}My Contributions{% endblock %}
{% block content %}
<h2>My Contributions</h2>
<div class="container">
  {% for c in contributions %}
```

```
  <div class="card mb-3">
    <div class="row no-gutters">
      <div class="col-md-8">
        <div class="card-body">
          <h5 class="card-title">{{ c.ObservationType }}</h5>
          <p class="card-text">{{ c.Report }}</p>
          <p class="card-text">
            <small class="text-muted">Date: {{ c.Date.strftime('%Y-%m-%d')
}}</small>
          </p>
        </div>
      </div>
      <div class="col-md-4">
        {% if c.Image %}
        <img src="{{ url_for('static', filename=c.Image) }}" class="card-
img" alt="Contribution Image">
        {% else %}
        <img src="{{ url_for('static',
filename='images/default_contribution.png') }}" class="card-img"
alt="Default Image">
        {% endif %}
      </div>
    </div>
  </div>
  {% endfor %}
</div>
{% endblock %}
```

Deletion of Account

delete_account.html

```
{% extends "base.html" %}
{% block title %}Delete Account{% endblock %}
{% block content %}
<div class="container mt-5">
  <h2>Confirm Account Deletion</h2>
  <p>Are you sure you want to delete your account? This action cannot be
undone.</p>
  <form method="POST">
    <button type="submit" class="btn btn-danger">Yes, Delete My
Account</button>
    <a href="{{ url_for('user_routes.dashboard') }}" class="btn btn-
secondary">Cancel</a>
  </form>
</div>
{% endblock %}
```

Home Page

Home.html

```
{% extends "base.html" %}
{% block title %}Home - Wildlife Conservation{% endblock %}
{% block content %}
```

```
<div class="jumbotron text-center">
  <h1>Welcome to Wildlife Conservation</h1>
  <p>Our mission is to protect and conserve wildlife and their
habitats.</p>
</div>
{% endblock %}
```

Sign In Page

signin.html

```
{% extends "base.html" %}
{% block title %}Sign In{% endblock %}
{% block content %}
<h2>Sign In</h2>
<form method="POST" action="">
  {{ form.hidden_tag() }}
  <div class="form-group">
    {{ form.email.label(class="form-label") }}
    {{ form.email(class="form-control") }}
  </div>
  <div class="form-group">
    {{ form.password.label(class="form-label") }}
    {{ form.password(class="form-control") }}
  </div>
  {{ form.submit(class="btn btn-primary") }}
</form>
{% endblock %}
```

Sign Up Page

signup.html

```
{% extends "base.html" %}
{% block title %}Sign Up{% endblock %}
{% block content %}
<h2>Sign Up</h2>
<form method="POST">
  {{ form.hidden_tag() }}
  <div class="form-group">
    {{ form.username.label(class="form-label") }}
    {{ form.username(class="form-control") }}
  </div>
  <div class="form-group">
    {{ form.email.label(class="form-label") }}
    {{ form.email(class="form-control") }}
  </div>
  <div class="form-group">
    {{ form.address.label(class="form-label") }}
    {{ form.address(class="form-control") }}
  </div>
  <div class="form-group">
    {{ form.password.label(class="form-label") }}
    {{ form.password(class="form-control") }}
  </div>
```

```
  <div class="form-group">
    {{ form.password2.label(class="form-label") }}
    {{ form.password2(class="form-control") }}
  </div>
  <div class="form-group">
    {{ form.community.label(class="form-label") }}
    {{ form.community(class="form-control") }}
  </div>
  {{ form.submit(class="btn btn-primary") }}
</form>
{% endblock %}
```

Display Species Page

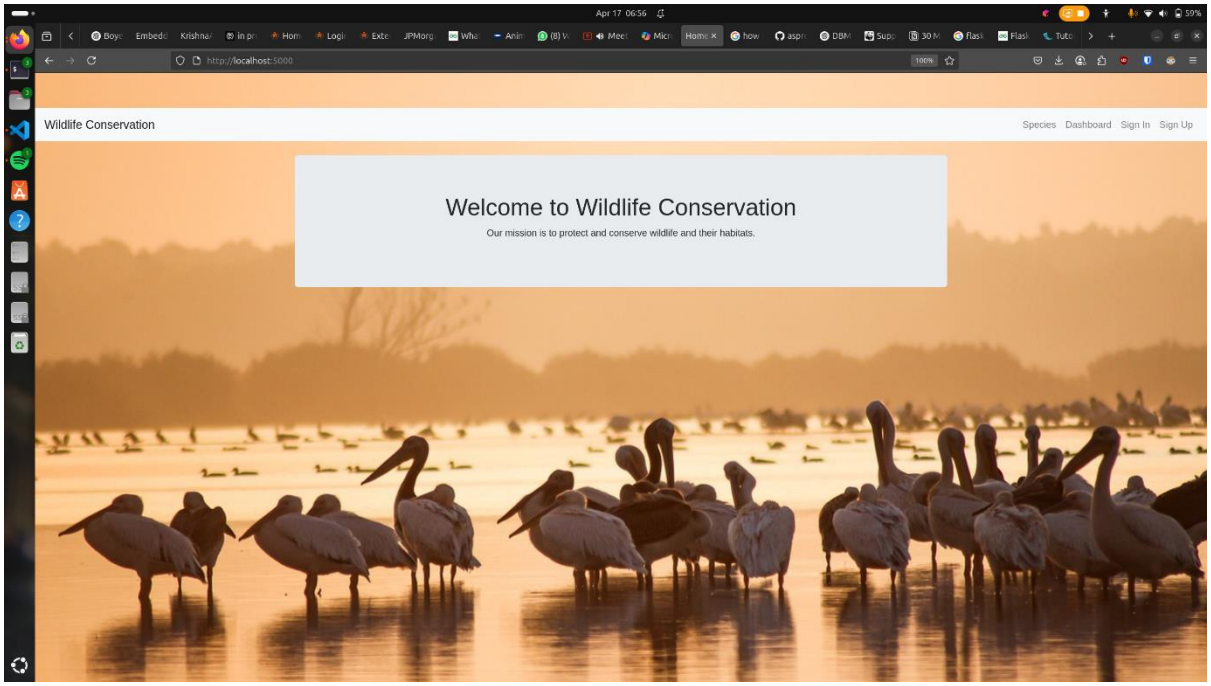
Species.html

```
{% extends "base.html" %}
{% block title %}Species{% endblock %}
{% block content %}
<div class="row">
  <div class="col-md-3">
    <h4>Filter By</h4>
    <form method="GET">
      <div class="form-group">
        <label for="habitat">Habitat</label>
        <input type="text" name="habitat" class="form-control">
      </div>
      <div class="form-group">
        <label for="region">Region</label>
        <input type="text" name="region" class="form-control">
      </div>
      <button type="submit" class="btn btn-primary">Filter</button>
    </form>
  </div>
  <div class="col-md-9">
    <h3>Species List</h3>
    <div class="row">
      {% for sp in species %}
      <div class="col-md-4">
        <div class="card mb-4">
          <!-- The image source is generated dynamically using
sp.Species_ID -->
          <img src="{{ url_for('static', filename='images/' ~ sp.Species_ID
~ '.jpg') }}" class="card-img-top" alt="{{ sp.EnglishName }}">
          <img src="{{ url_for('static', filename='images/' ~ sp.Species_ID
~ '.jpeg') }}" class="card-img-top" alt="{{ sp.EnglishName }}">
          <div class="card-body">
            <h5 class="card-title">{{ sp.EnglishName }}</h5>
            <p class="card-text">{{ sp.LocalName }}</p>
            <a href="#" class="btn btn-primary">Details</a>
          </div>
        </div>
      </div>
      {% endfor %}
```

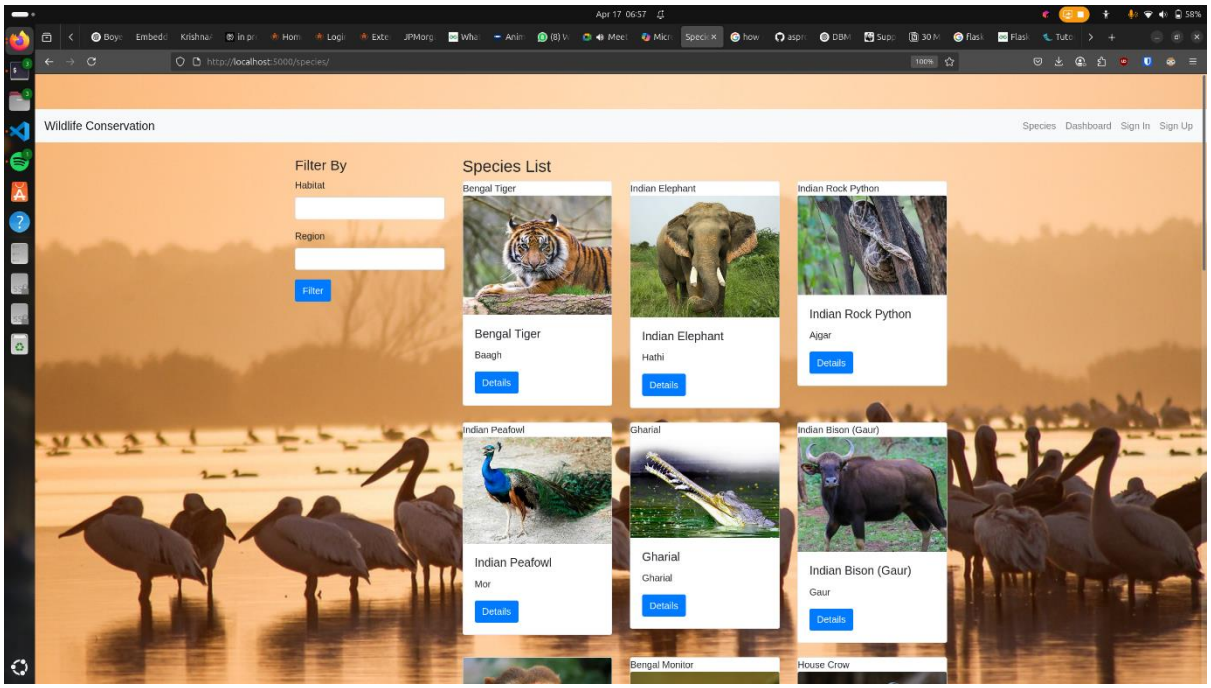
```
    </div>
  </div>
</div>
{% endblock %}
```

UI Screenshots

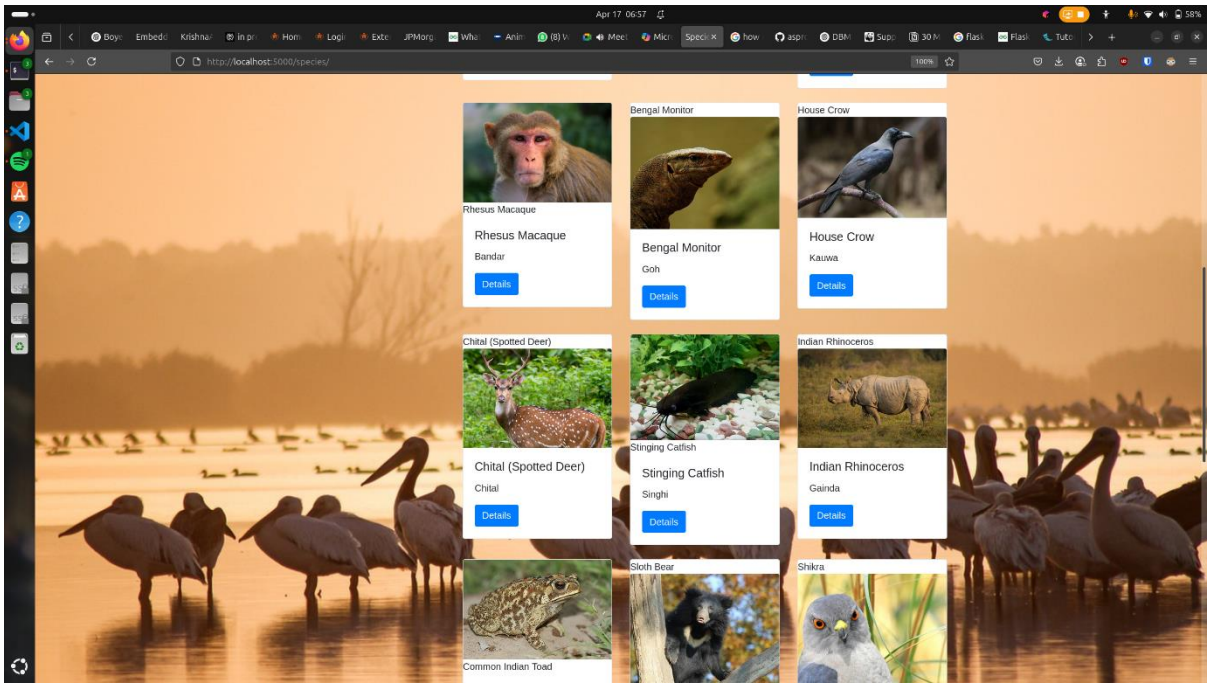
Home Page



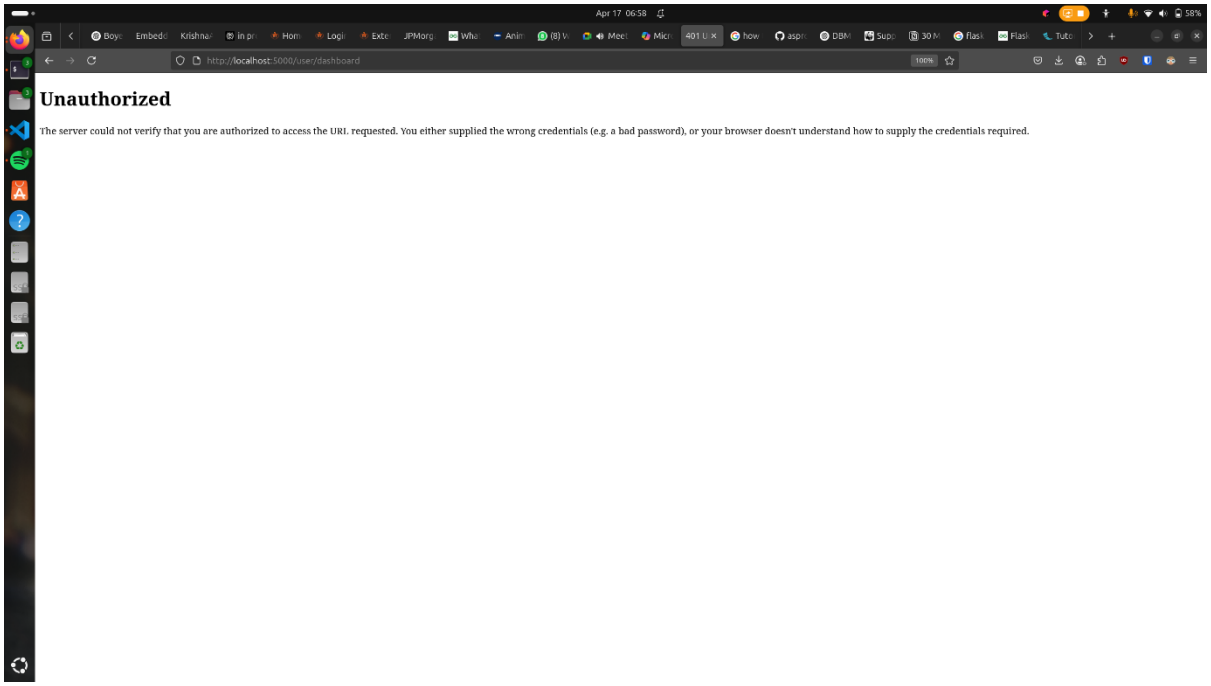
Species Tab – displays all species in database



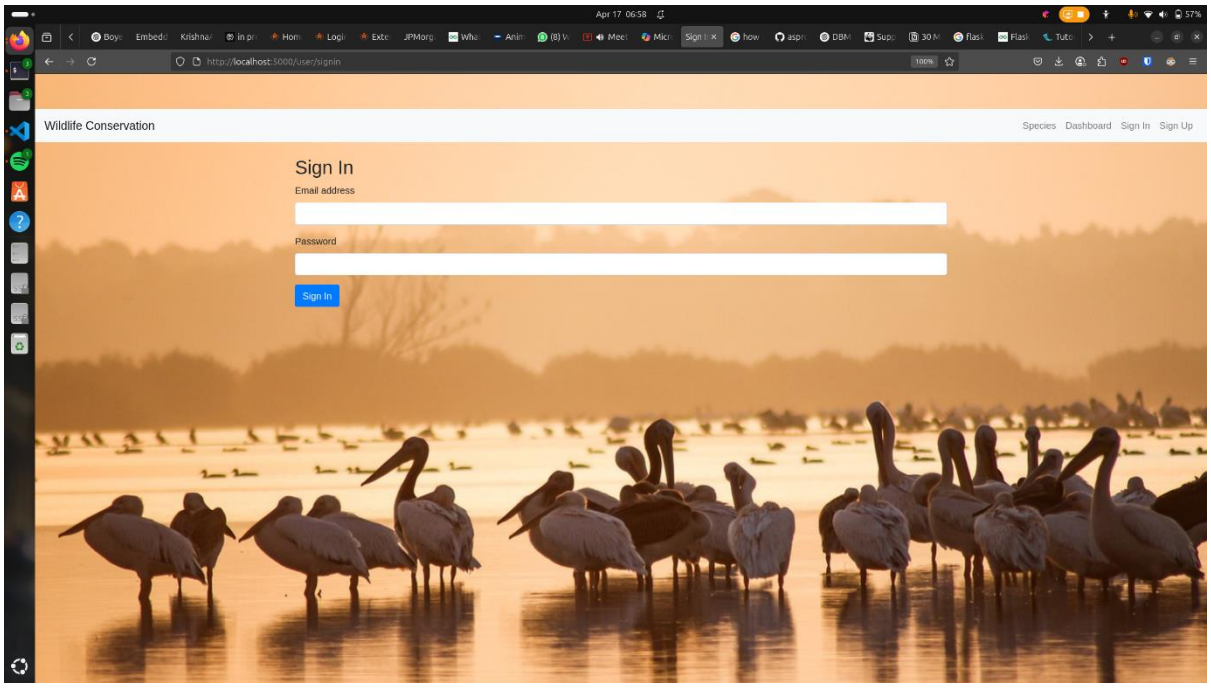
Species Display Scrolled Down



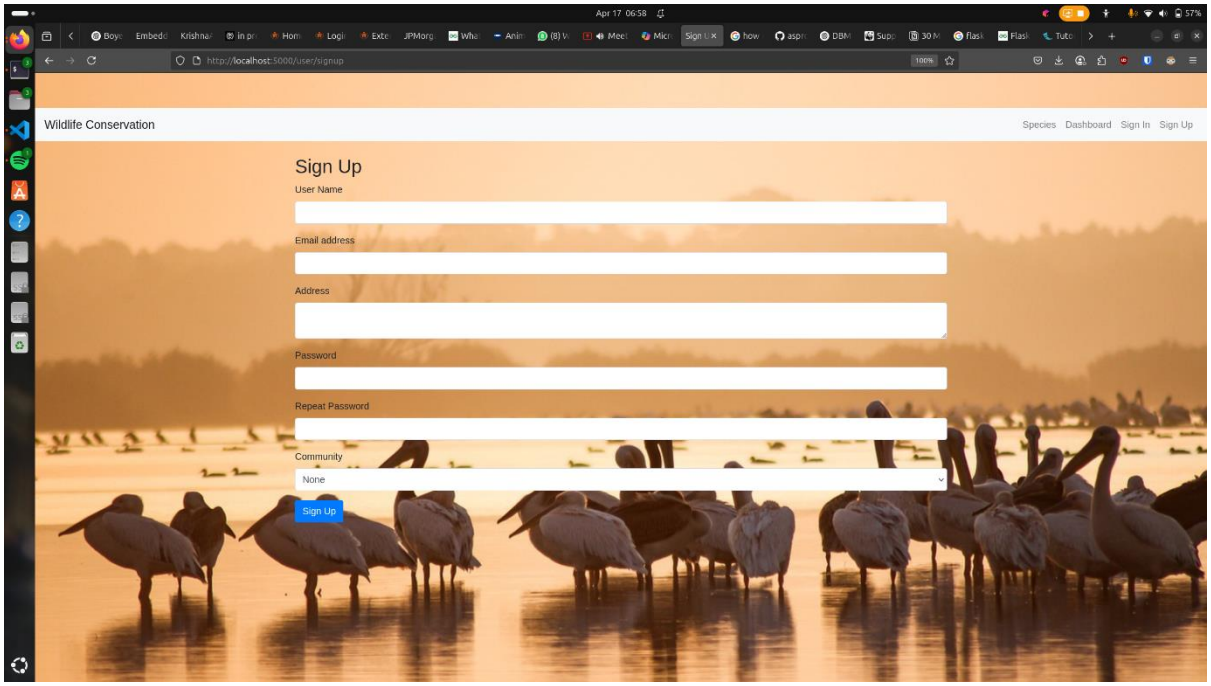
Unauthorised Accession of the Dashboard – trying to access dashboard without an account



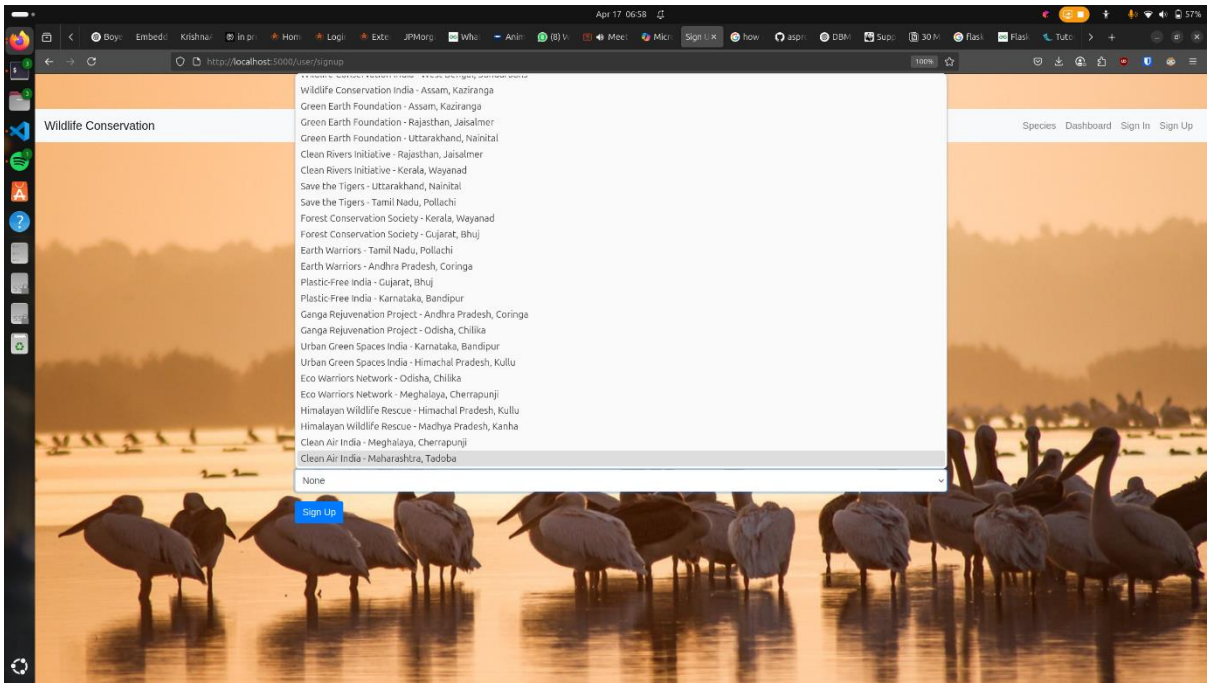
Sign-In Page



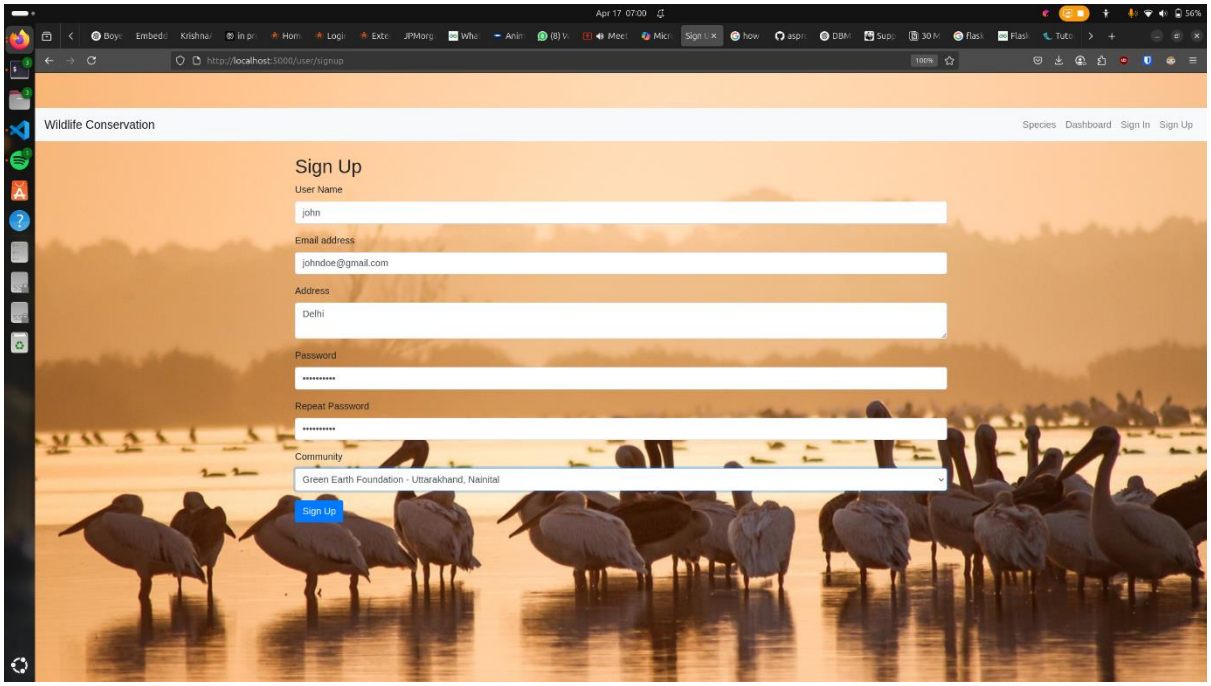
Sign-Up Page



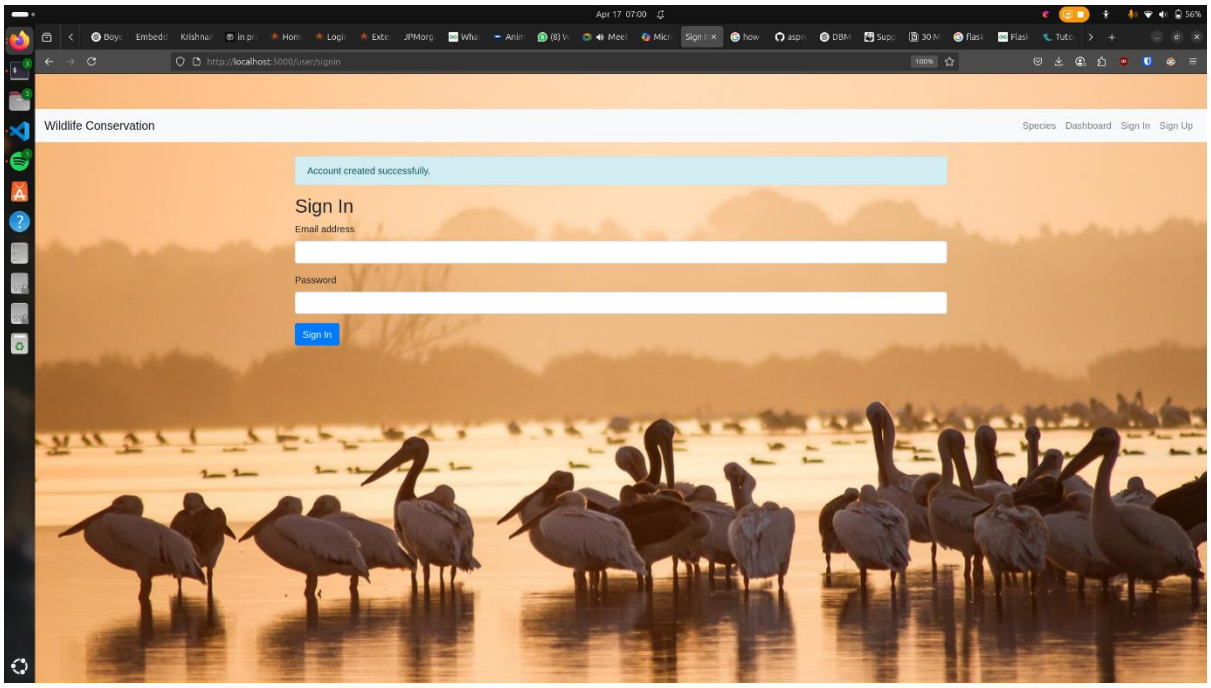
Drop-down on Sign-Up Page to Choose from Communities Registered on Portal



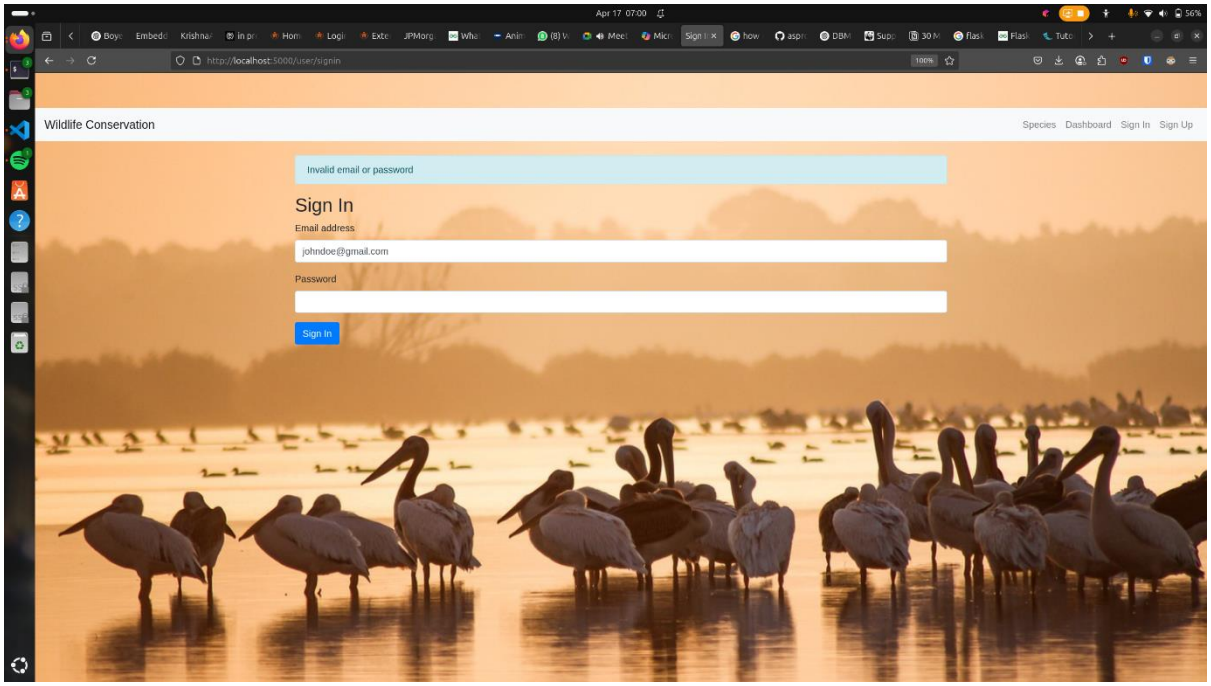
Example User Sign-Up



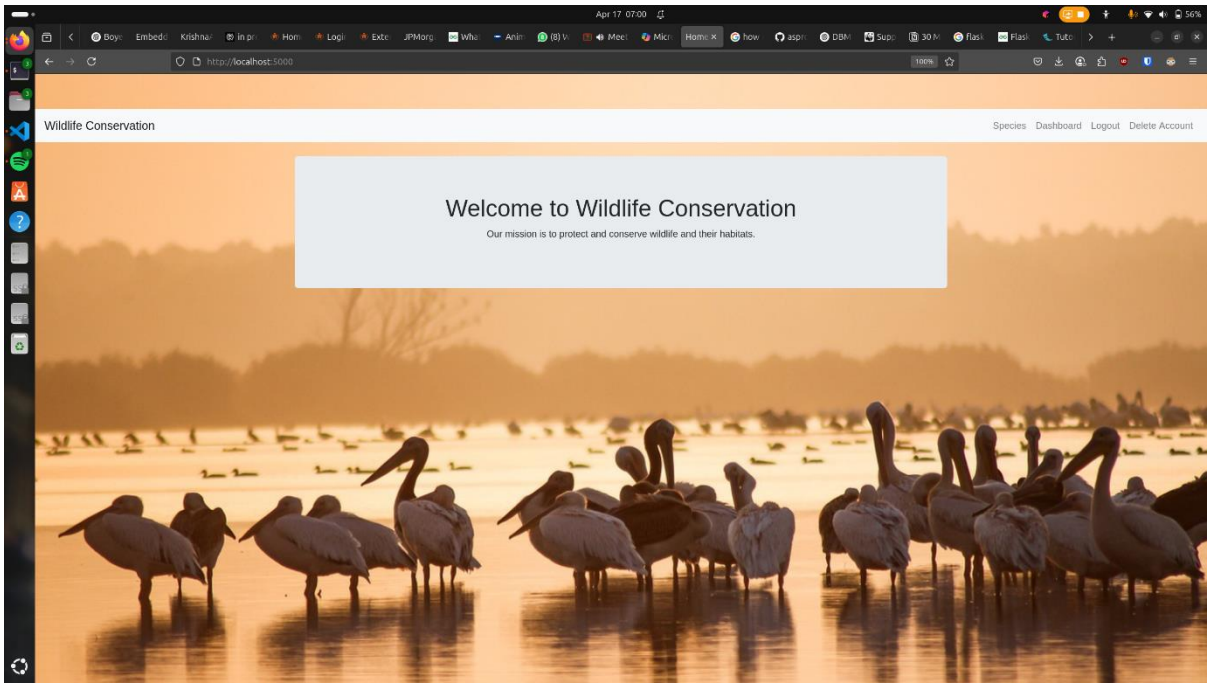
Account Created Message



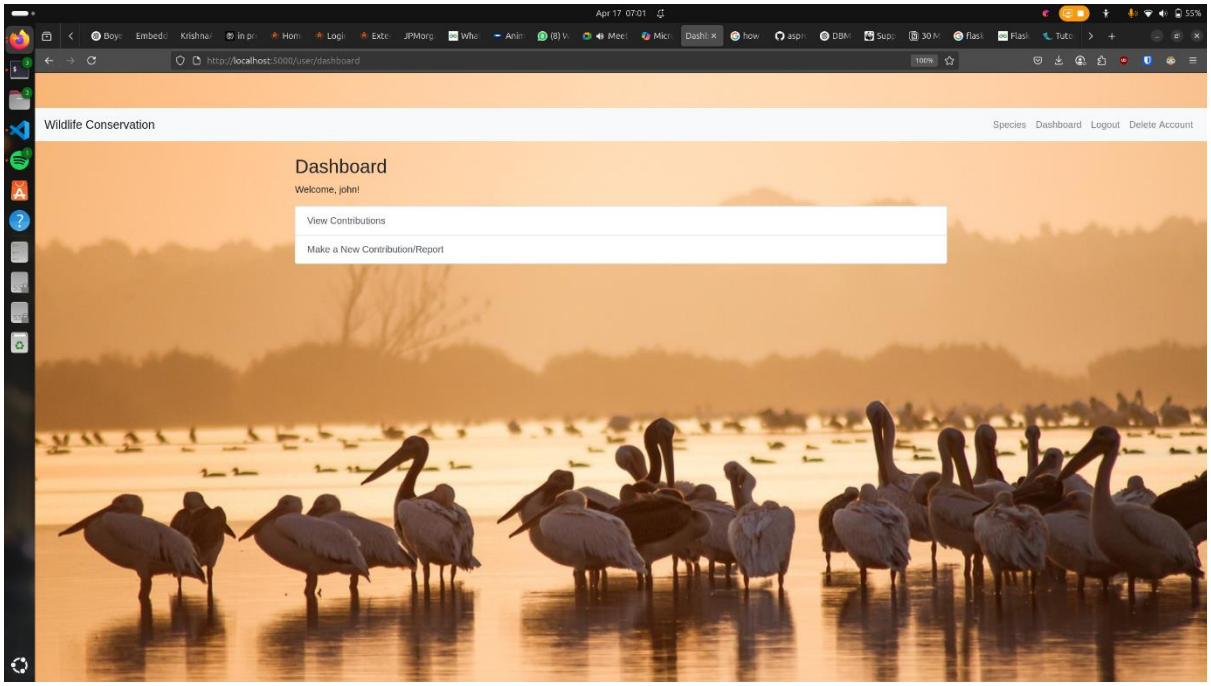
Invalid ID/Password while Signing-in



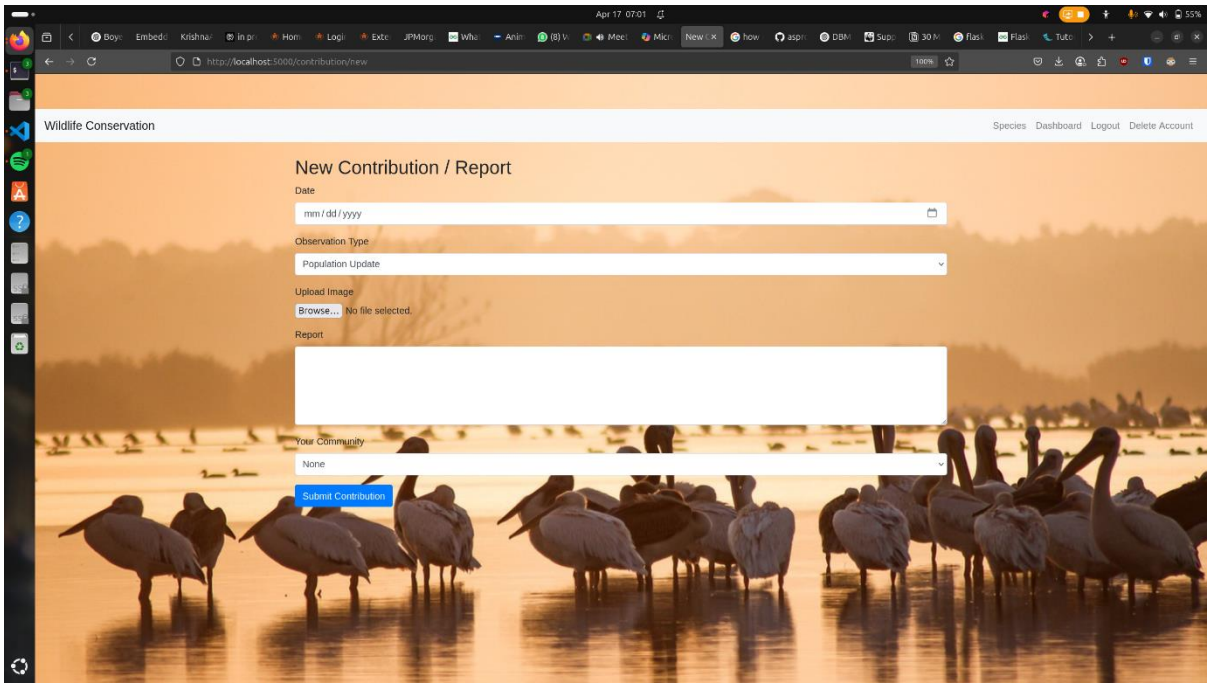
Welcome Page



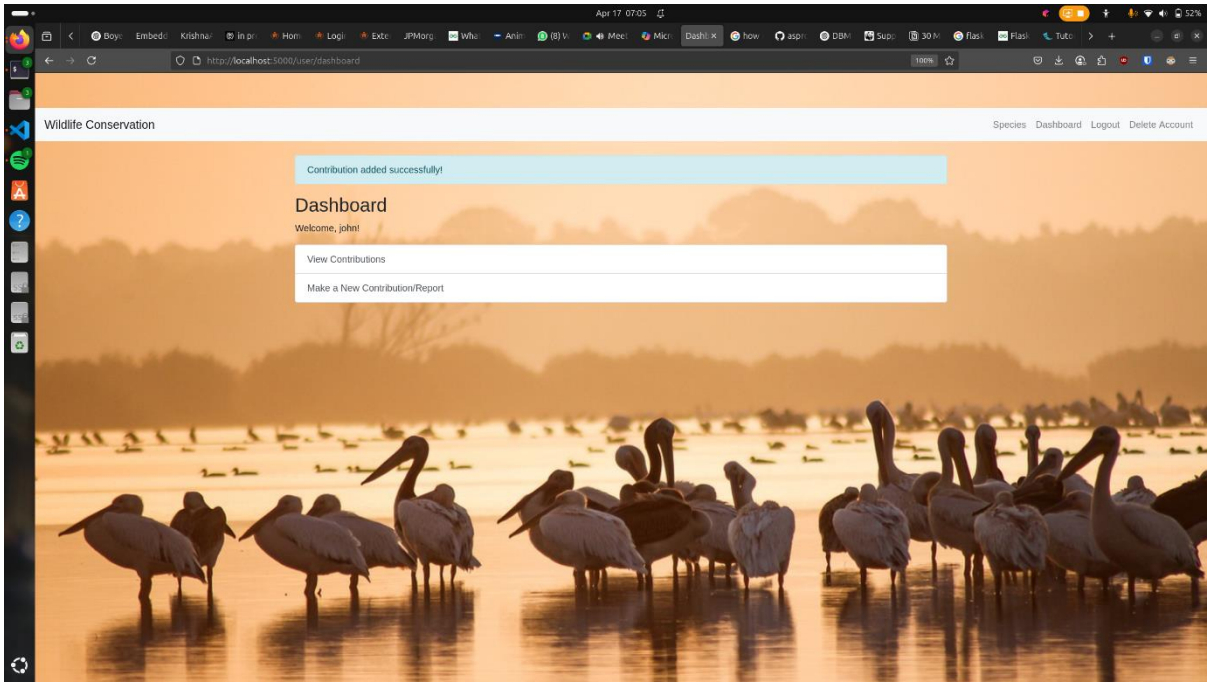
Dashboard



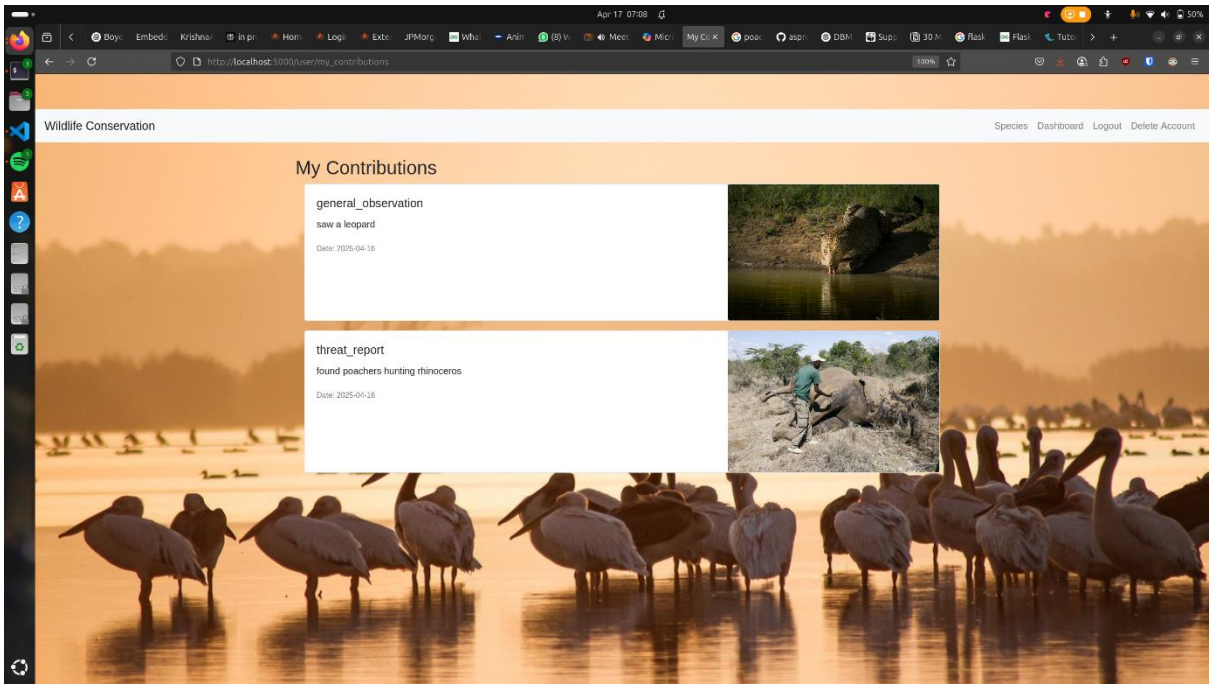
Creating a New Contribution



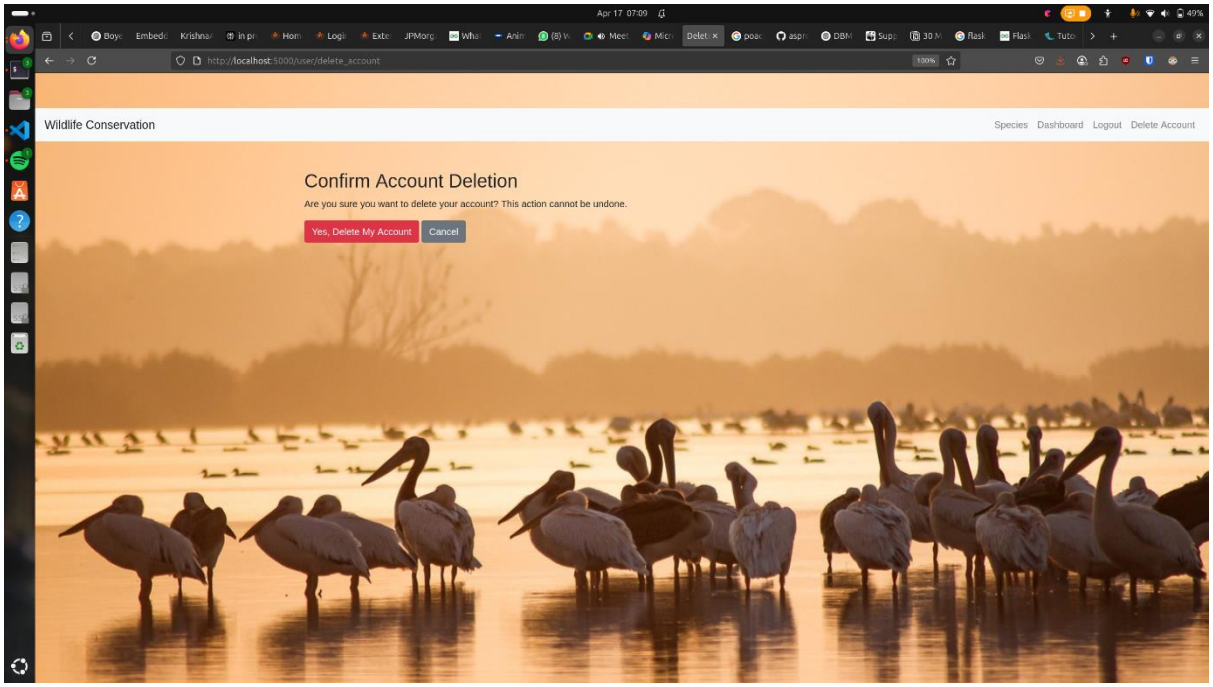
Contribution Added Message



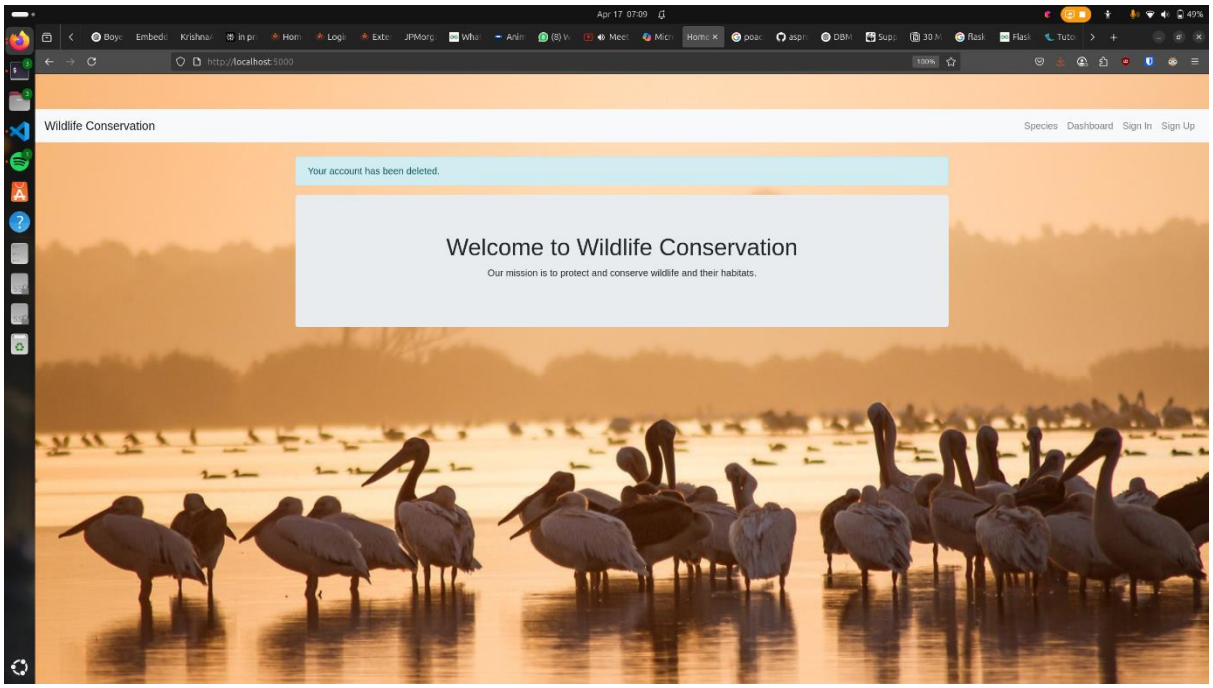
View Contributions



Deletion of an account



Account deleted message



References

Flask Documentation: https://flask.palletsprojects.com/
MySQL Docs: https://dev.mysql.com/doc/
SQLAlchemy ORM: https://docs.sqlalchemy.org/
Bootstrap for UI Styling: https://getbootstrap.com/

Other references for learning:
https://w3schools.com/
https://geeksforgeeks.com/
https://tutorialspoint.com/

Video References:
Codemy – YouTube channel