CSC 490 2024 Fall Final Report

WildGuard

Project Team: Meghan Wilcox, Derek Fox, William Harper, Alex Jasper

Table of Contents

1. Project Definition (300-500 words)

- **Why (it is needed)**
  - Conservation efforts are becoming increasingly critical as endangered species face growing threats, including habitat destruction, climate change, and poaching. Current platforms, such as iNaturalist, while helpful for general biodiversity tracking, lack features specifically tailored for endangered species conservation. These limitations create a need for a platform that addresses the unique challenges of data collection, analysis, and collaboration in this area. WildGuard seeks to bridge this gap by providing a specialized web application to support and enhance conservation efforts, ensuring vital data on endangered species is effectively utilized.
- **What (is the goal of the project)**
  - The primary objective of WildGuard is to develop a role-based platform that facilitates real-time reporting, predictive analytics, and secure data management to prioritize conservation efforts. The platform aims to centralize crucial data, such as poaching incidents and population trends, to enable informed decision-making. A key feature of WildGuard is its crowdsourcing capability, which allows volunteers to contribute valuable data through sightings and observations. This not only enriches the available information but also strengthens machine learning algorithms used to analyze trends and predict species population changes. By addressing the challenges of tracking elusive species, WildGuard seeks to empower researchers, conservationists, and communities with actionable insights to protect endangered species effectively.
- **How (how will it be achieved)**
  - WildGuard leverages cutting-edge web technologies to deliver an intuitive and robust platform. The frontend is built with React.js, ensuring a responsive and user-friendly interface, while the backend utilizes Django for scalability and security. The platform incorporates machine learning frameworks like TensorFlow and Keras to enable predictive analytics for population trends and threats. Additionally, geospatial mapping tools provide a visual representation of species distributions and habitat changes. Role-specific access controls ensure that data is securely managed, with tailored permissions for researchers, volunteers, and administrators. By combining modern technology with collaborative features, WildGuard creates a comprehensive tool to support conservation priorities and improve endangered species outcomes.

- **Teamwork Division**
  - **Meghan Wilcox**: Designed and implemented the database, developed the homepage and navigation bar user interface, developed the backend, implemented the map and wildlife sighting components, integrated the machine learning model into the user interface, and implemented user permissions throughout the website.
  - **Alex Jasper**: Developed and Co-developed site wide UI/UX with Meghan, secure and persistent session management, expanded database functionality for users, user profile, quality assurance testing and fixing many frontend and backend bugs along the way.
  - **Billy Harper**: Developed and implemented animal population prediction algorithms.
  - **Derek Fox**: Designed and trained the machine learning image recognition feature, as well as integrated the finished model into the backend of the app.
- **Source Code (GitHub Repository Link)**
  - https://github.com/meghanwilcox/CSC490Capstone-UNCG-2024
- **Demo (Video Link)**
  - https://www.youtube.com/watch?v=yozXpm-AdVM

2. Project Requirements Analysis

- **Functional**
  - Role-based access for researchers and volunteers.
  - Machine learning integration for predictive analytics.
  - Access to educational materials.
  - Interactive geospatial map for sightings and analytics.
  - Easy to use, minimalist UI/UX design and implementation
- **Usability**
  - User interface
    - Minimalist design for user-friendly interaction.
    - Data entry forms with validation.
    - Universal navbar across all pages for easy navigation to any part of the website.
- **System**
  - **Software**
    - Frontend
      - Built using React.js with the Vite build tool for fast development and optimized builds.
      - Incorporates interactive maps with Leaflet and React-Leaflet.
      - Routing handled by React Router DOM.
      - HTTP requests managed with Axios.
    - Backend:
      - Developed with Django, leveraging Django REST Framework for API implementation.
      - Includes middleware for Cross-Origin Resource Sharing (CORS) configuration.
    - Database:
      - Azure MySQL is used to store species sightings and user-related data.
      - Data types include species classifications, images, user email, hashed passwords (encrypted using Argon2), and user roles (e.g., admin, researcher, volunteer).
    - ML:
      - TensorFlow and Keras are utilized for training and deploying an animal classification model.
      - The model classifies species based on a dataset of 90 animal classes.
      - Jupyter Notebook is used for training and experimentation.
      - TensorFlow is GPU-optimized for faster computations using CUDA.

    - **Database**
        - MySQL DBSM hosted on Azure for storing species sightings data and other user related data (e.g. email, name, hashed password, role, etc.).
- **Security**
    - **Authentication**:
        - Passwords are securely hashed using Argon2, a cryptographic hash function resistant to attacks.
        - Django handles secure user authentication workflows.
    - **Role-Based Access Control (RBAC)**:
        - Role management ensures only authorized users can perform specific actions (e.g., admins manage users, researchers submit sightings).
        - Implements access constraints to safeguard sensitive data and prevent unauthorized edits or retrievals.

3. Project Specification

- **Focus / Domain / Area**
  - Serves as a way to help researchers and wildlife conservationist have a more centralized means of sharing, collecting and learning from data pertaining to endangered species with the functionality to encompass more than just endangered and critically endangered.
- **Libraries / Frameworks / Development Environment**
  - Frontend: React.js with Vite for development and React-Leaflet for map integration.
  - Backend: Django with Django REST Framework for APIs.
  - Database: MySQL hosted on Azure with SSL encryption.
  - ML Frameworks: TensorFlow and Keras for training and deployment, supported by Pandas for data preprocessing and Matplotlib for visualizations.
  - IDE: Visual Studio Code for frontend, PyCharm for backend, and JupyterLab for ML experimentation.
- **Platform (Mobile, Desktop, Gaming, Etc)**
  - WildGuard is a desktop web application designed for browsers. Future developments will focus on mobile platforms to simplify photo and data uploads, enabling on-the-go usage for field researchers.
- **Genre (Game, Application, etc)**
  - Wildlife Monitoring and Education Web Application
- **Dataset / Data Resources**
  - **Dataset Source:**
    - The machine learning model is trained using the Animal Image Dataset (90 Different Animals) from Kaggle.
  - **Dataset Details:**
    - Classes: 90 animal classes, including both common and exotic species.
    - Images: Over 32,000 images, organized into folders by class, each containing labeled examples for supervised training.
    - Format: Images in .jpg format, with directories structured for direct use in ML workflows.
  - **Usage in Project:**
    - The dataset is used to train a TensorFlow/Keras model for species classification.
    - Preprocessing includes resizing, normalization, and data augmentation to improve model generalization.
  - **Integration with System:**
    - The trained model is deployed in the Django backend to classify user-uploaded animal images.

- User uploads and predictions are stored in the Azure MySQL database to build a repository of classified images.

4. System Design

- **Overall System Architecture**
  - The system architecture is divided into three main components: the frontend, backend, and database, with a machine learning model integrated into the backend for predictive analytics. The application adheres to the Model-View-Controller (MVC) architecture for efficient separation of concerns and maintainability.

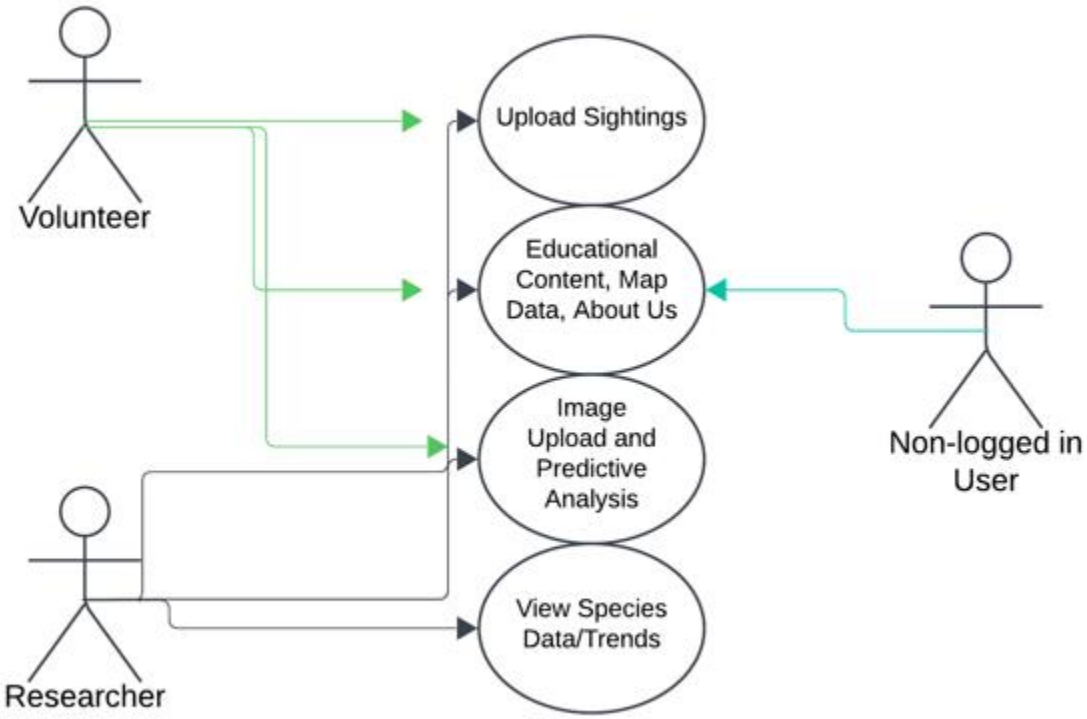

Figure 1: Use Case Diagram illustrating the interactions of volunteers, researchers, and non-logged-in users with key functionalities such as uploading sightings, accessing educational content, uploading images for predictive analysis, and viewing species data or trends.

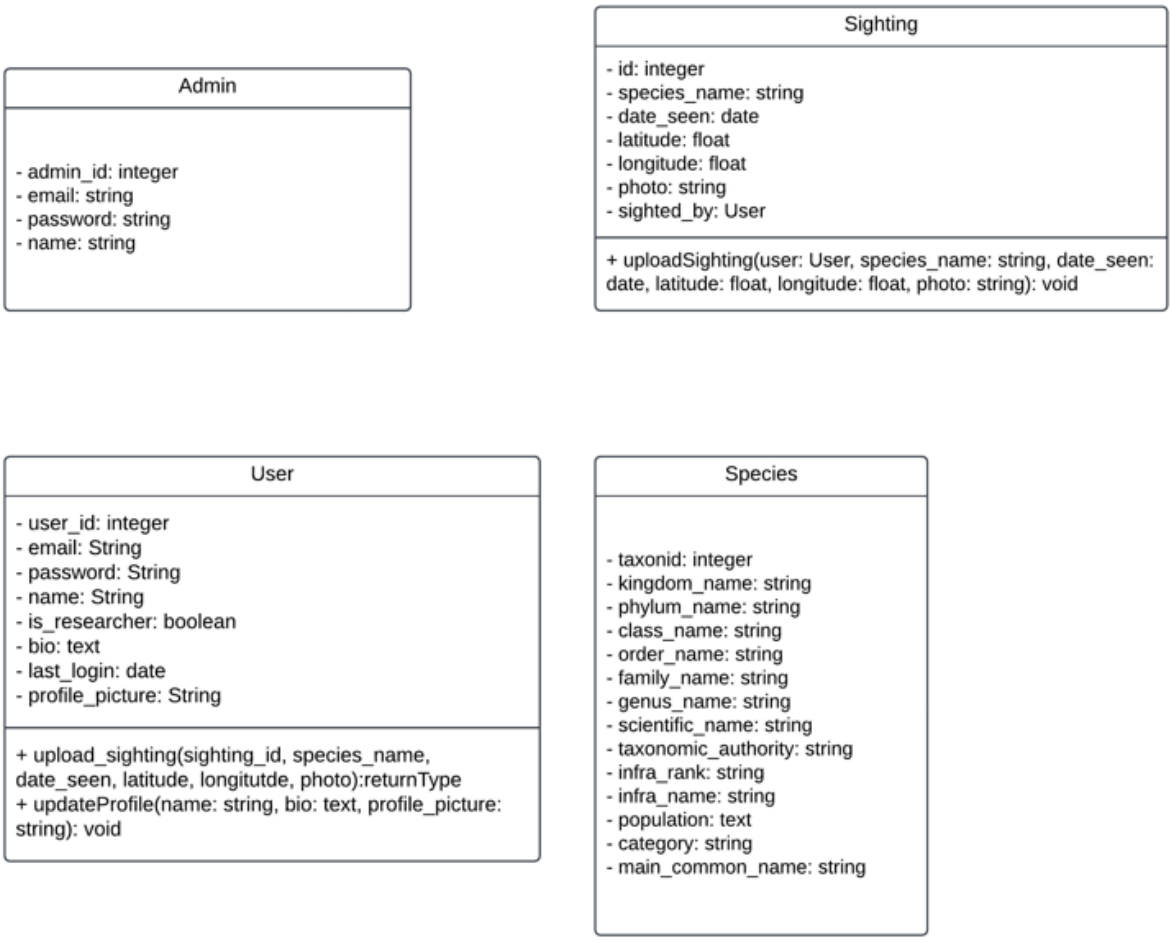

Figure 2: UML class diagram showcases the relationships and attributes of the Admin, User, Sighting, and Species entities, highlighting their key properties and methods for a wildlife management system

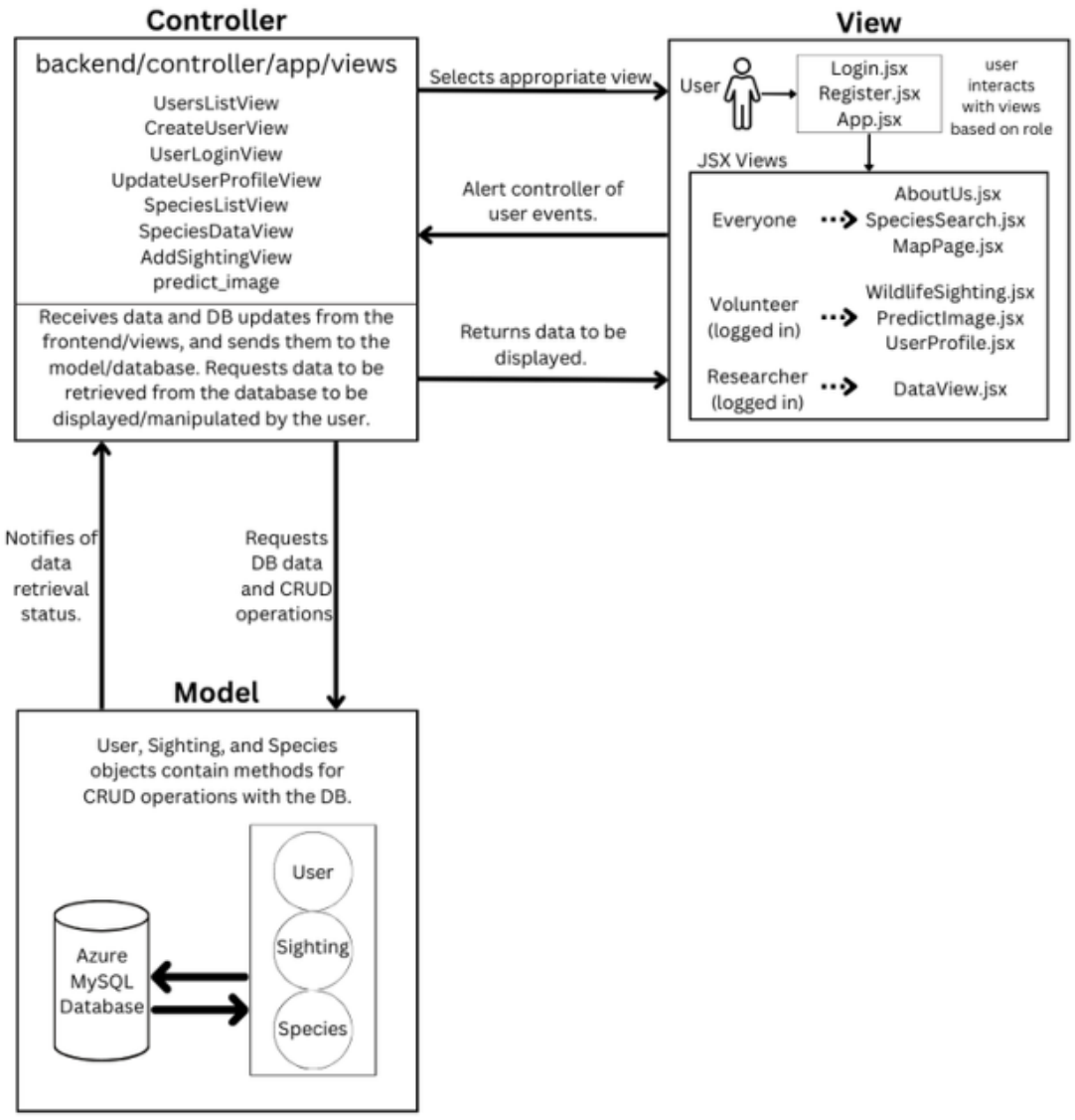

Figure 3: A high-level UML diagram depicting the Model-View-Controller (MVC) architecture of the WildGuard application, illustrating the interactions between the user interface, backend controllers, and database models.

Subsystems

1. **Subsystem1: Frontend**
    - Developed using React.js with Vite for fast builds.
    - Incorporates React-Leaflet for map visualizations.
    - Communicates with the backend via HTTP using axios.
2. **Subsystem2: Backend**
    - Built with Django and Django REST Framework.
    - Implements role-based access control and CSRF protection.
3. **Subsystem3: Database**
    - MySQL database hosted on Azure.
    - Stores user credentials (hashed passwords), roles, and species reporting data.
4. **Subsystem4: Machine Learning Model**
    - TensorFlow/Keras-based classifier trained on a Kaggle dataset of 90 animal classes.
    - Outputs species predictions to assist users in identifying sightings.

5. Evaluation

Typical Usage Examples

1. **User Registration and Login**:
    - A new user registers via the React frontend, and the backend validates and stores the data in the database.
    - Upon login, the user's role determines access permissions (e.g., admin can view all reports).
2. **Species Reporting**:
    - A user uploads an image of an animal and fills out the form which obtains specifics details such as latitude and longitude, time/date seen, and suspected name of animal. The React app sends the form data to the backend.
    - The backend processes the image with the ML model and returns a species prediction.
    - The prediction and image are stored in the database for future reference.
3. **Interactive Map**:
    - Users can view reported species sighting, poaching data, and filter for specific results on an interactive map using Leaflet.

System Performance

- **Frontend**:
    - React with Vite ensures fast UI rendering and quick development builds.
- **Backend**:
    - Handles typical API requests in under 500ms for basic CRUD operations.
    - ML model inference time: ~1-2 seconds per image.
- **Database**:
    - Optimized for fast read/write operations for up to 1000 concurrent users.
    - Uses indexes for efficient querying of species data.

6. Discussion

- **Limitations and Future Work**
  - **Limitations**:
    - **Basic ML Model**: The current model is limited to 90 animal classes and may not generalize well to unseen data.
    - **Scalability**: The system has not been stress-tested for high user loads or large datasets.
- **Future Work**:
  - **Production HTTPS**: Use a trusted SSL certificate for production deployment.
  - **Improved ML Model**: Retrain the model with a larger, more diverse dataset and optimize for better performance.
  - **Scalability**: Deploy the backend on a scalable cloud platform (e.g., AWS, Azure App Service). Also implement a mobile app version for a more streamlined UX and easier data transfer.
- **Other Thoughts**
  - The integration of ML for species classification adds significant value for researchers and wildlife enthusiasts.
  - Ensuring a smooth user experience on both frontend and backend has been a key focus, but further usability testing is recommended.