QUT College

Assessment Cover Sheet

Students

Student name	Student number
Aidan Iliave Cakaukeivuya	11865156
Gyeonghwan “Jack” Noh	12200263
Marimbay Kadirov	12198226

Class

Tutor	Geoff Polzin
Tutorial Day, Time & Room	Thursday, 11:00 AM, P229

Video Demonstration Link

https://www.youtube.com/shorts/6HNOlb4xO-k

GitHub Repository Link

https://github.com/Marimbay/novenom.git

Statement of Contribution

Student 1 Aidan	Flask backend, routing, file upload system, documentation, project coordination, writing report, drawing system diagram. ~30%
Student 2 Gyeonghwan	Full front-end (HTML/CSS), responsive design, UI polish, demo video editing, writing report. ~30%
Student 3 Marimbay	AI server integration (Hugging Face + PyTorch), SQLite DB, GitHub, Cursor AI dev/refactor codebase, writing report, uploading demo video on YouTube. ~40%

Project Objectives

To use a Raspberry Pi to coordinate an input-output system for detecting venomous creatures. By processing a photo of an animal from the user's camera with a locally-ran artificial intelligence model to analyse whether or not the creature is venomous.

To increase complexity of the project: one, connecting all group members to GitHub using GIT for easy collaboration and backup; two, improving the appearance and understanding of the website; and three, connecting a database for history tracking of photos analysed.

Review and Discussion of Technologies Used

Hardware

1. Raspberry Pi

The Raspberry Pi is a single-board computer that serves as an ideal platform for this project due to its low cost and energy efficiency (Mathe et al., 2024). It is the technology required to be built around for the project and is covered throughout ITD102. It is used to host the website which coordinates the entire system. There are many specialised and more powerful alternatives like the NVIDIA Jetson though the Raspberry Pi was provided and configured properly by the unit.

2. Laptop

A laptop is a personal computer that is portable (GCFGlobal.org, n.d.). It is used to host the webservice and stores the AI models locally. It is also used to provide a reliable development and testing environment compared to the Raspberry Pi because it takes steps to setup as well as its dependency on the QUT Wi-Fi. Cloud servers could be used as an alternative however a laptop is easy-to-setup and less internet-dependency.

3. Phone Camera

A smartphone is computer which aimed allow a telephone to have other features such as an operating system, web browser and camera (Kirvan & Provazza, 2023). Specifically, the camera, this is used as the primary input device for capturing quality images of creatures for analysis and can be connected to a web browser. Webcams and cameras could be used as alternatives, but phone cameras provide the convenience of being readily available and easily integrated with the web application.

Front-end Technologies

4. Web Browser

The web browser is a software application that allows one to access and view websites on a network (BrowserStack, 2025). It serves as the primary interface for users to interact with the application, rendering the HTML, Tailwind CSS and certain Flask libraries, allowing users to view and interact with the website. While a mobile app could be developed as an alternative, a web browser provides the advantage of cross-platform compatibility without requiring installation.

5. HTML

Hypertext Markup Language, HTML, forms the structural foundation of the web application. It defines the elements and their relationships on the page, creating the framework that CSS styles and JavaScript enhances. While alternatives like XML exist for specific use cases, HTML remains the standard for web content structure.

6. Tailwind CSS

Tailwind CSS is a utility-first CSS framework that accelerates the development of the user interface (Fitzgerald, 2022). It is used to provide pre-built classes for styling, allowing for rapid UI development without writing custom CSS. Frameworks like Bootstrap or Material-UI offer similar functionality, but Tailwind's utility-first approach provides more flexibility and smaller bundle sizes.

7. Awesome Font

Awesome Font provides a comprehensive library of vector icons that enhance the user interface (WP Engine, 2023). These icons scale well across different screen sizes and resolutions, maintaining clarity at any size. Alternatives like Material Icons exist, but Awesome Font offers a wide variety of icons and consistent styling.

Back-end Technologies

8. Python

Python is the primary programming language used in this project. Its simplicity and extensive library ecosystem make it ideal for both web development and AI implementation (Scarlett, 2023). Alternatives like Node.js or Java could be used, but Python's readability and the availability of AI libraries make it the preferred choice.

9. Flask

Flask is a lightweight web framework that provides the foundation for the application's backend (Dyouri & Walia, 2024). It routes HTTP requests to Python functions, handling the web server interface. While more comprehensive frameworks like Django exist, Flask's flexibility and simplicity make it suitable for this project's needs.

10. Werkzeug Utilities (werkzeug.utils)

Werkzeug provides the WSGI utilities that interface between the web server and the Flask application (Kennedy, 2024). Specifically, Werkzeug Utilities and its secure_filename function, this allows a file, i.e. the uploaded image, to be securely transferred and stored onto the website. Alternatives include Gunicorn though Werkzeug is usually well integrated with Flask, making it the natural choice.

11. SQLite

SQLite is a lightweight, file-based database that stores the application's data (Sinyaniso, 2025). Its simplicity and zero-configuration setup make it ideal for this project. More powerful alternative databases like PostgreSQL exist, but SQLite's file-based nature and ease of use make it suitable for the application's needs.

12. At Exit (atexit)

The At Exit module ensures proper cleanup of resources when the application terminates (McKee, 2024). It registers functions to run on program exit, preventing resource leaks. For the project, used to delete all uploaded images upon exit. Alternatives like context managers or try-finally blocks could be used though At Exit provides a centralized way to handle cleanup.

Utilities

13. Shell Utilities (shutil)

The shutil module provides high-level file operations for managing the application's files (Python Software Foundation, n.d.). It handles tasks like file copying, moving, and deletion. This is used with other modules with editing uploaded images to be stored correctly. Alternatives like the os module provides similar functionality, but shutil offers more convenient high-level operations.

14. Universally Unique Identifier (uuid)

The uuid module generates unique identifiers for files and database records. It creates 128-bit unique identifiers that are virtually guaranteed to be unique (Python Software Foundation, n.d.). This is used to stop a bug of uploading different images with the same name. Traditional alternatives of using timestamp-based or sequential IDs could be used though UUIDs provide better uniqueness guarantees.

15. Pathlib

Pathlib provides an object-oriented interface to file system paths (Python Software Foundation, n.d.). It handles path manipulation in a cross-platform manner, ensuring compatibility across different operating systems. While os.path provides similar functionality, Pathlib’s object-oriented approach is more intuitive.

16. Requests

The Requests library simplifies HTTP communication between the application and external services (Reitz et al., 2023). It provides a user-friendly interface for making HTTP requests. While urllib is available in the standard library, Requests offers a more intuitive API.

17. Base64

Base64 encoding converts binary data (like images) into ASCII text for transmission (Kumar, 2023). It ensures that binary data can be safely transferred over text-based protocols. While alternatives like hex encoding exist, Base64 is the standard for this purpose.

AI Integration

In summary, PyTorch is used to communicate with and for the AI model while Hugging Face Transformers and ResNet-50 combine to make the AI models.

18. PyTorch

PyTorch is a deep learning framework that provides the tools for implementing and running neural networks (Chen et al., 2019). It handles tensor operations and automatic differentiation, making it easier to implement complex AI models. While TensorFlow is a popular alternative, PyTorch's dynamic computation graph and Pythonic interface make it more intuitive for many developers.

19. Hugging Face Transformers

Hugging Face Transformers provides access to pre-trained models for various AI tasks (Wolf et al., 2020). It offers a wide range of model architectures and weights, making it easy to implement state-of-the-art AI solutions. While TensorFlow Hub offers similar functionality, Hugging Face's focus on transformers and ease of use makes it the preferred choice.

20. ResNet-50

ResNet-50 is a deep neural network architecture specifically designed for image classification (He et al., 2016). Its residual learning approach allows for training very deep networks effectively. Alternatives like VGG or Inception exist, but ResNet-50's balance of accuracy and efficiency makes it suitable for this project.

21. Python Pillow (PIL)

Pillow is a comprehensive image processing library that handles image manipulation tasks (Clark, 2015). It is particularly used for translating images of various formats to be understandable for the AI. Alternatives like OpenCV offers more advanced computer vision and image breakdown capabilities, but Pillow's simplicity and Python integration make it ideal for basic image processing tasks.

22. I/O (io)

The I/O module provides the tools for handling data streams in the application. It manages the flow of data between different parts of the system (Python Software Foundation, n.d.). It is used in the AI integration to provide temporary memory for processing. Traditional alternatives like file operations or socket programming could be used directly, the I/O module provides a more abstract and convenient interface.

AI Assistants

A number of AI assistants were used to generating code and assisting it technology implementation and debugging. Multiple AIs were used because of each group member’s preference at the time though they are all truthful and good co-developers.

23. Cursor AI

Cursor AI is an AI-powered code editor that provides intelligent code assistance (Cursor, 2025). It offers code suggestions and completions, helping developers write code more efficiently. While alternatives like GitHub Copilot exist, Cursor AI's integration with the development environment makes it particularly useful.

24. ChatGPT

ChatGPT is a large language model that provides natural language understanding and generation capabilities (Bansal et al., 2024). It can assist with various tasks, from answering questions to generating code. While alternatives like Claude or Bard exist, ChatGPT's widespread adoption and capabilities make it a popular choice.

25. Google Gemini

Google Gemini is a multimodal AI model that can process both text and images. It offers advanced understanding and generation capabilities across multiple modalities (Gemini Team et al., 2023). While alternatives like GPT-4 exist, Gemini's multimodal capabilities make it particularly useful for applications involving both text and images.

Version Control

26. GitHub and GIT

GitHub is a code hosting platform that provides version control and collaboration features (GitHub, 2023). It hosts GIT repositories and offers tools for code review, issue tracking, and project management. It works with GIT and is used for version control, code collaboration across multiple devices and provides a backup. Alternatives like GitLab or Bitbucket exist, but GitHub's widespread adoption and feature set make it the standard for many projects.

Design and Implementation

Note that AI assistants were used to generate code and help we installation steps throughout. Use one, its instructions should look similar to the following steps.

All technologies were implemented in **5 major sections** including prerequisites:

0. Prerequisites

Before implementing, ensure that the Raspberry Pi is set up properly. These were all done in tutorials. You can refer to the setup document found on Canvas.

Ensure that the Raspberry Pi is:

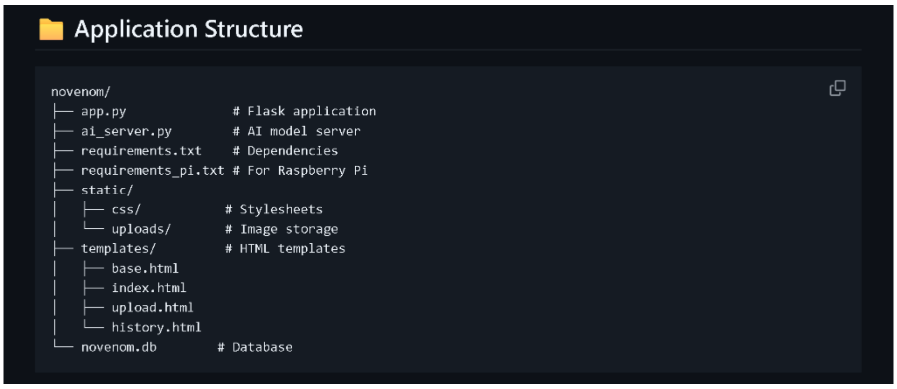
☑ Running Raspberry Pi OS
☑ Connected to the QUT internet (important when hosting)
☑ Up-to-date

1. **Python Flask** and **Werkzeug Utilities**

a. Ensure Python is up-to-date then install Flask and Werkzeug. Enter the following code:

```
sudo apt install python3 python3-pip -y
pip3 install flask
pip3 install --upgrade flask werkzeug
```

Following the folder structure in the image below, create the directories by entering link the code following:



Application Structure

```
novenom/
├── app.py              # Flask application
├── ai_server.py        # AI model server
├── requirements.txt    # Dependencies
├── requirements_pi.txt # For Raspberry Pi
├── static/
│   ├── css/           # Stylesheets
│   └── uploads/       # Image storage
├── templates/         # HTML templates
│   ├── base.html
│   ├── index.html
│   ├── upload.html
│   └── history.html
└── novenom.db        # Database
```

```
mkdir -p my_flask_app/static/uploads
mkdir -p my_flask_app/templates
cd my_flask_app
```

b. Create the website file app.py using nano.

For an example of the code, refer to the GitHub link at cover page.

An important note is to replace the IP address (0.0.0.0) at the end of the code with the Raspberry Pi's IP address which was used to connect to it through Putty. Alternatively, you can find the Raspberry Pi's IP address by entering this code in the Pi terminal:

```
hostname -I
```

2. **Version Control System** (GitHub and GIT)

a. Learn Git commands with Google Gemini and implement version control system (VCS) to the project.

b. Create an account on GitHub (if not already) and create a repository https://github.com/Marimbay/novenom

c. Connect ssh of Raspberry Pi to repository. For this task we used following commands:

```
# Generate a new SSH key pair
ssh-keygen -t ed25519 -C "email@example.com"

# Start SSH agent
eval "$(ssh-agent -s)"

# Add private key
ssh-add ~/.ssh/id_ed25519

# To see and copy your public key
cat ~/.ssh/id_ed25519.pub

# Add your public key to GitHub account
Go to GitHub Settings > SSH and GPG keys > New SSH key. Paste copied public key here.

# Test SSH connection to Github
ssh –T git@github.com

# Update repository's remote URL to use SSH (if it was HTTPS)
git remote –v #shows current remote URL
git remote set-url origin git@github.com:Marimbay/novenom.git
```

d. Standard git workflow commands (for daily operations)

```
# Provides description about Git’s current state
git status

# Branches allow us to experiment with new versions and modifications without affecting current
working progress
        1. In order to switch to the branch
git checkout (branch name e.g. main)
        2. In order to create new local branch
git checkout –b (name of new branch e.g. test)

# After making changes to the files we need to stage and commit them as well as fetch changes that was
made.
   1. In order to stage all the changes, we use git add . or git add (name of modified file e.g.
novenom.db) to stage specific files.
   2. In order to commit we use git commit –m “Concise commit message e.g. Update filename etc.”
   3. In order to publish changes so that everyone can see and access them we use git push command
   4. In order to get latest changes from the remote we use git pull
```

3. HTML and CSS

There are three frontend templates

- Read file: templates/base.html
- Read file: templates/index.html
- Read file: templates/upload.html

Here is a step-by-step:

a. Base Template (base.html)

The base template is the main layout for other pages. It contains:

A1. CSS Framework and Dependencies

These codes load Tailwind CSS for quick styling utilities and icons to use easy.

```
<!-- Tailwind CSS for styling -->
<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
<!-- Font Awesome for icons -->
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
```

A2. Custom CSS Classes

These are the codes for reusable classes. We use btn for the padding, rounded corners. Flash adds margin, transition. Card creates white background, rounded box and etc.

```
/* Custom button styles */
.btn {
  @apply px-4 py-2 rounded-lg transition-all duration-200 ease-in-out;
}
.btn-primary {
  @apply bg-green-600 text-white hover:bg-green-700;
}

/* Flash message styles */
.flash-message {
  @apply p-4 mb-4 rounded-lg transition-all duration-300;
}

/* Card styles */
.card {@apply bg-white rounded-xl shadow-lg p-6 transition-all duration-
```

A3. Navigation

Nav creates a white shadow and his three navigation links

```
<nav class="bg-white shadow-lg">
  <!-- Logo -->
  <a href="{{ url_for('index') }}" class="text-2xl font-bold text-green-600">
    <i class="fas fa-spider mr-2"></i>NoVenom
  </a>

  <!-- Navigation Links -->
  <a href="{{ url_for('index') }}">Home</a>
  <a href="{{ url_for('upload_image') }}">Upload</a>
  <a href="{{ url_for('history') }}">History</a>
</nav>
```

b. Home page (index.html)

The home page is divided into several sections:

B1. Hero section

This codese explains about descriptive paragraph, and a styled button that links to upload page.

```
<div class="text-center py-12">
  <h1 class="text-4xl font-bold text-gray-900 mb-4">
    Identify Venomous Creatures with AI
  </h1>
  <p class="text-xl text-gray-600">
    Upload a photo of any insect or small creature...
  </p>
  <a href="{{ url_for('upload_image') }}" class="btn btn-primary">
    <i class="fas fa-upload mr-2"></i>Upload Photo
  </a>
</div>
```

B2. Feature Grid

This code creates grid inlcudes icon, heading, and a short description.

```
<div class="grid md:grid-cols-3 gap-8">
  <!-- Three feature cards -->
  <div class="card">
    <i class="fas fa-camera text-4xl text-green-600"></i>
    <h3>Easy Upload</h3>
    <p>Simply take a photo and upload it...</p>
  </div>
  <!-- More feature cards... -->
</div>
```

B3. How it works Section

This code is showing a numbered circle, a heading, and a short description.

```
<div class="grid md:grid-cols-4 gap-8">
  <!-- Four steps with numbered circles -->
  <div class="text-center">
    <div class="bg-green-100 rounded-full w-16 h-16">
      <span class="text-2xl font-bold text-green-600">1</span>
    </div>
    <h3>Take Photo</h3>
    <p>Capture a clear image...</p>
  </div>
  <!-- More steps... -->
</div>
```

c. Upload Page (upload.html)

The upload page is divided into two sections:

C1. Upload Form

This code lets user to put and upload an image.

```
<form action="{{ url_for('upload_image') }}" method="post" enctype="multipart/form-data">
  <!-- Drag and drop zone -->
  <div class="border-2 border-dashed border-gray-300 rounded-lg p-8" id="drop-zone">
    <input type="file" name="file" id="file-input" class="hidden" accept="image/*">
    <!-- Upload interface -->
  </div>

  <!-- Submit button -->
  <button type="submit" class="btn btn-primary" id="submit-btn" disabled>
    <i class="fas fa-search mr-2"></i>Analyze Image
  </button>
</form>
```

C2. Results Display

This code includes uploaded images on the left and venomous status box and a progress bar based on percentage on the right.

```
<div class="card mt-8">
  <div class="grid md:grid-cols-2 gap-8">
    <!-- Image display -->
    <div>
      <img src="{{ image_url }}" alt="Uploaded creature" class="rounded-lg">
    </div>

    <!-- Results -->
    <div class="space-y-4">
      <!-- Venom status -->
      <div class="p-4 rounded-lg {% if result == 'Venomous' %}bg-red-100{% else %}bg-green-100{% endif
%}">
        <p class="text-xl font-bold">{{ result }}</p>
      </div>

      <!-- Confidence bar -->
      <div class="w-full bg-gray-200 rounded-full h-2.5">
        <div class="bg-green-600 h-2.5 rounded-full w-[{{ confidence }}%]"></div>
      </div>
    </div>
  </div>
</div>
```

d. JavaScript Functionality

This code includes drag-and-drop events, and adds a green border.

```
// File input handling
fileInput.addEventListener('change', function() {
  if (this.files.length > 0) {
    submitBtn.disabled = false;
    updateDropZoneText(this.files[0].name);
  }
});

// Drag and drop handling
['dragenter', 'dragover', 'dragleave', 'drop'].forEach(eventName => {
  dropZone.addEventListener(eventName, preventDefaults, false);
});

// Visual feedback for drag and drop
function highlight(e) {
  dropZone.classList.add('border-green-500');
}
```

e. Responsive design

The frontend is fully responsive using Tailwind's responsive classes:

- `md:grid-cols-3`: 3 columns on medium-sized screens and larger.
- `max-w-7xl`: The container won't get wider.
- `text-4xl`: text are big sizes that adjust for readability on each device.
- Mobile-friendly navigation with hamburger menu

f. Error Handling

This code includes flashed messages inside a div, using red and green styling for errors and success.

```
{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
    {% for category, message in messages %}
      <div class="flash-message {% if category == 'error' %}flash-error{% else %}flash-success{% endif
%}">
        {{ message }}
      </div>
    {% endfor %}
  {% endif %}
{% endwith %}
```

4. Hugging Face transformers, PyTorch and ResNet-50

1. Model Initialization process

```
from transformers import AutoImageProcessor, AutoModelForImageClassification

MODEL_NAME = "microsoft/resnet-50"
processor = AutoImageProcessor.from_pretrained(MODEL_NAME)
model = AutoModelForImageClassification.from_pretrained(MODEL_NAME)
```

2. Image processing pipeline

2.1 Image reception

```
image_data = request.json['image']
image_bytes = base64.b64decode(image_data)
image = Image.open(io.BytesIO(image_bytes))
```

2.2 Preprocessing

Transforms image for model input

```
inputs = processor(images=image, return_tensors="pt")
```

outputs PyTorch tensor with shape [1, 3, 224, 224]

3. Prediction process

3.1 Model inference

Runs inference

```
with torch.no_grad():
    outputs = model(**inputs)
    logits = outputs.logits
    probabilities = torch.nn.functional.softmax(logits, dim=-1)
```

3.2 Result processing

```
# Get top prediction
top_prob, top_class = torch.max(probabilities, dim=1)

# Check for venomous creatures
venomous_classes = ['spider', 'snake', 'scorpion', 'wasp', 'bee']
class_name = model.config.id2label[top_class.item()]
is_venomous = any(venomous in class_name.lower() for venomous in venomous_classes)
```

4. Response format

```
return {
‘is_venomous’: is_venomous,
‘confidence’: float(top_prob.item()),
‘class_name’: class_name
}
```

5. Integration points of ai_server.py and app.py

5.1 Main app to AI server

in app.py

```
def analyze_image(image_path):
    with open(image_path, 'rb') as image_file:
        image_data = base64.b64encode(image_file.read()).decode('utf-8')
    response = requests.post(AI_SERVER_URL, json={'image': image_data})
    return response.json()
```

5.2 AI server to model

In ai_server.py

```
@app.route('/analyze', methods=['POST'])
def analyze():
    image_data = request.json['image']
    result = analyze_image(image_data)
    return jsonify(result)
```

5. SQLite

5.1 Database configuration

```
# Database path configuration
DATABASE_PATH = os.path.join(app.root_path, 'novenom.db')

# Database initialization
def init_db():
    conn = sqlite3.connect(DATABASE_PATH)
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS predictions (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            filename TEXT NOT NULL,
            timestamp DATETIME NOT NULL,
            is_venomous BOOLEAN NOT NULL,
            animal_name TEXT,
            confidence REAL,
            image_path TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()
```

5.2 Database operations

Connection management

```
def get_db():
    """Get a database connection with row factory"""
    conn = sqlite3.connect(DATABASE_PATH)
    conn.row_factory = sqlite3.Row
    return conn
```

Saving predictions

```
def save_prediction(filename, is_venomous, animal_name, confidence, image_url):
    try:
        conn = get_db()
        c = conn.cursor()
        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')

        # Clean up animal name
        animal_name = clean_species_name(animal_name)

        c.execute('''
            INSERT INTO predictions
            (filename, timestamp, is_venomous, animal_name, confidence, image_path)
            VALUES (?, ?, ?, ?, ?, ?)
        ''', (filename, current_time, is_venomous, animal_name, confidence, image_url))
        conn.commit()
    except Exception as e:
        app.logger.error(f"Error saving prediction: {str(e)}")
    finally:
        conn.close()
```

Retrieving history

```
@app.route('/history')
def history():
    try:
        conn = get_db()
        c = conn.cursor()
        predictions = c.execute('''
            SELECT * FROM predictions
            ORDER BY timestamp DESC
            LIMIT 10
        ''').fetchall()

        # Convert Row objects to dictionaries
        predictions_list = []
        for pred in predictions:
            pred_dict = dict(pred)
            if isinstance(pred_dict['timestamp'], str):
                pred_dict['timestamp'] = datetime.strptime(
                    pred_dict['timestamp'],
                    '%Y-%m-%d %H:%M:%S.%f'
                )
            predictions_list.append(pred_dict)

        return render_template('history.html', predictions=predictions_list)
    except Exception as e:
        app.logger.error(f"Error fetching history: {str(e)}")
        return redirect(url_for('index'))
```

5.2 Database maintenance

Cleanup function

```
def cleanup():
    """Clean up database and upload folder when the app exits."""
    try:
        # Clear the database
        conn = sqlite3.connect(DATABASE_PATH)
        c = conn.cursor()
        c.execute('DELETE FROM predictions')
        conn.commit()
        conn.close()
    except Exception as e:
        app.logger.error(f"Error during cleanup: {str(e)}")
```

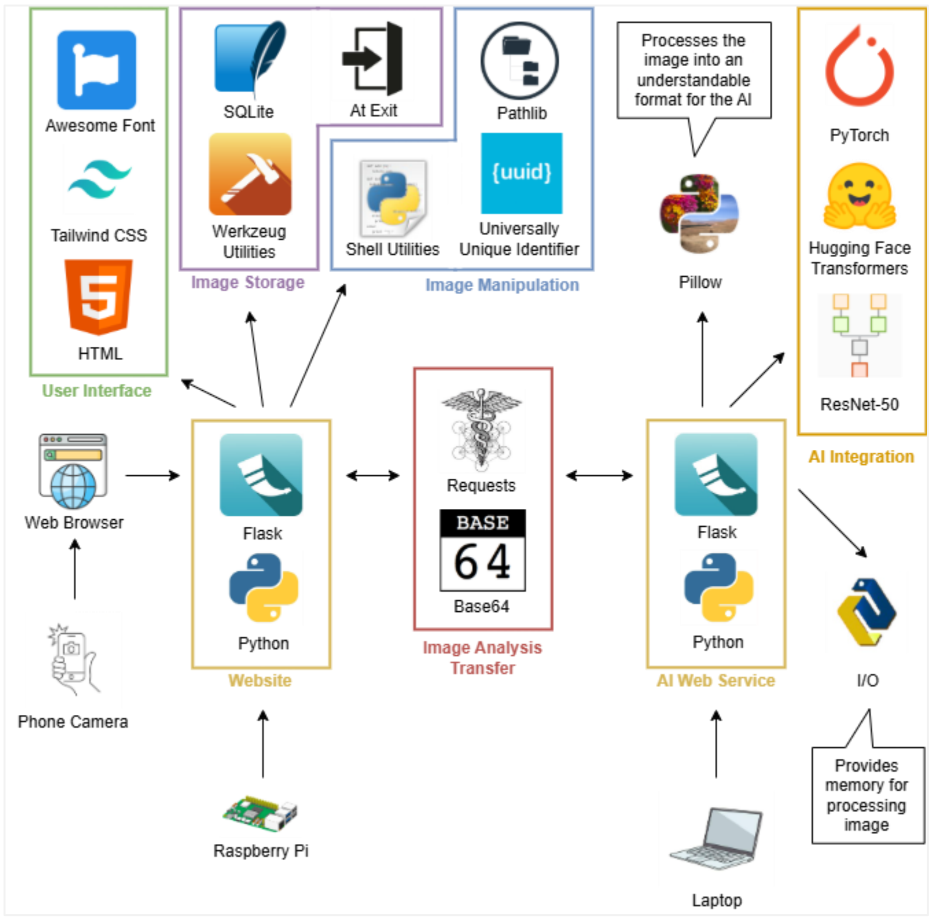
Data formatting for user-friendly layout

```
def clean_species_name(name):
    """Clean and format the species name."""
    if not name:
        return "Unknown Species"

    # Split by comma and take the first part
    name = name.split(',')[0].strip()

    # Capitalize each word
    name = ' '.join(word.capitalize() for word in name.split())

    return name
```

How It Works

Novenom is a web app designed to recognize venomous creatures using uploaded images. A user simply goes to the website, and either takes a photo, or selects a file containing the image of the creature. The system processes the image using AI, analyses, and checks whether the creature is venomous. The result will be displayed with color-coded indicators: red indicating venomous, green indicating non-venomous, and showing the species along with the confidence level of the analysis. All users can keep track of their previous analyses in the history section, which keeps a log of all uploads, results, and timestamps. Everything can be accessed from mobile and desktop, which makes the workflow seamless and accessible for anyone who wishes to quickly and efficiently identify potentially dangerous creatures.

System Diagram



Challenges We Faced

While we were working on the Novenom project, we met multiple challenges of considerable importance. Such challenges required complex decision-making considerations that greatly impacted project refinement. Here are the critical problems we encountered and the solutions we derived:

1. Management of the dependency's versions

The very first problem we encountered had to do with the debugging complications related to the compatibility of versioning of different libraries. Such problems had an impact on the overall stability of the application. In response to this, we created an organized methodology that dealt with dependency problems. Initially, we executed the **pip install** command for all the required packages, followed by the generation of a dependency list using the **pip freeze** command. We later created the **requirements.txt** and **requirements_pi.txt** files for utilization by **ai_server.py** and **app.py** respectively. This approach ensured consistent behaviour across different development environments and eliminated version-related bugs that were causing instability in the application.

2. Firewall Configuration

A major problem happened when the website and web service were not connecting, which affected the application's functionality. Our reconciliation process started by generating simple test requests aimed toward isolating the problem. It became very clear very fast that the absence of responses points towards some form of networking issue. After pinpointing the network firewall as the most likely cause, we undertook an initial exploration of disabling the firewall in order to validate our hypothesis. This allowed us to implement proper exceptions to the requests on the firewall, then re-enable the firewall with working security policies.

3. User Interface Modernization

We discovered that our project's CSS styling practices (implemented at the begging) significantly impacted user experience in a negative way. We began solving this challenge by analyzing the existing interface, which resulted in identifying several limitations - there was too much unnecessary information possibly misleading user. Recognizing the need for a more user-centric design approach, we implemented a more minimalistic and professional UI/UX. This modernization effort involved adopting modern CSS practices and frameworks.

Future Directions and Improvements

Our Novenom project has numerous areas for improvements and expansions, which include:

1. **Switch to a better AI model**

Images are currently classified using the ResNet-50 model which has only 1000 classes of different objects and it can be significantly improved by switching to OpenAI's vision model, that offers prompted image classification requests simplifying logistics and improving accuracy of outputs.

2. **More file extensions**

In a bid to enhance the flexibility of the application, we plan to broaden its input scope beyond static photographs to include audio files for identifying venomous creatures by their sounds, video files for real time analysis, and geolocation data for situational context-awareness. Moreover, we intend to supports even more image formats so that users do not have to convert images beforehand.

3. **Turning into mobile app**

One of the most important future development areas includes changing the existing web-based platform into a dedicated mobile application. Such a transformation would improve user accessibility and experience as well as possibilites for use in offline modes in poor internet coverage regions. The application would retain all current functionalities with some enhancements, that would help users to identify venomous creatures in real-life situations. Such improvements would enhance the application's functionality, flexibility, and ease of use for users, allowing for easier identification of venomous creatures.

References

Bansal, G., Chamola, V., Hussain, A., Guizani, M., & Niyato, D. (2024). Transforming conversations with AI—A comprehensive study of ChatGPT. *Cognitive Computation, 16*(1), 2487–2510. https://doi.org/10.1007/s12559-023-10236-2

BrowserStack. (2025). *What is a Browser? How does it Work?* BrowserStack. https://www.browserstack.com/guide/what-is-browser/

Chen, K. M., Cofer, E. M., Zhou, J., & Troyanskaya, O. G. (2019). Selene: A PyTorch-based deep learning library for sequence data. *Nature Methods, 16*(4), 315–318. https://doi.org/10.1038/s41592-019-0360-8

Clark, A. (2015). *Pillow (PIL Fork) Documentation*. Python Imaging Library. https://pillow.readthedocs.io/en/stable/

Cursor. (2025). *Cursor (code editor)*. Wikipedia. https://en.wikipedia.org/wiki/Cursor_(code_editor)

Dyouri, A., & Walia, A. S. (2024) *How to Build a Flask Python Web Application from Scratch.* DigitalOcean. https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3

Fitzgerald, A. (2022). *Tailwind CSS: What It Is, Why Use It & Examples.* HubSpot. https://blog.hubspot.com/website/what-is-tailwind-css

GCFGlobal.org. (n.d.). *What is a laptop computer?* GCFGlobal.org. https://edu.gcfglobal.org/en/computerbasics/laptop-computers/1/

Gemini Team, Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., Silver, D., Johnson, M., Antonoglou, I., Schrittwieser, J., Glaese, A., Chen, J., Pitler, E., Lillicrap, T., Lazaridou, A., Firat, O., … Vinyals, O. (2023). *Gemini: A family of highly capable multimodal models*. arXiv. https://arxiv.org/abs/2312.11805

GitHub. (2023). *About GitHub and Git*. https://docs.github.com/en/get-started/start-your-journey/about-github-and-git

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Kennedy, P. (2024). *What is Werkzeug?* TestDriven.io. https://testdriven.io/blog/what-is-werkzeug/

Kirvan, P., & Provazza, A. (2023). *What is a Smartphone?* TechTarget. https://www.techtarget.com/searchmobilecomputing/definition/smartphone

Kumar, A. (2023, March 15). *Base64 Encoding: What Is It? How Does It Work?* Built In. https://builtin.com/software-engineering-perspectives/base64-encoding

Mathe, S. E., Kondaveeti, H. K., Vappangi, S., Vanambathina, S. D., & Kumaravelu, N. K. (2024). A comprehensive review on applications of Raspberry Pi. *Computer Science Review, 52*, 1-33. https://doi.org/10.1016/j.cosrev.2024.100636

McKee, A. (2024). *How to Exit Python: A Quick Tutorial.* DataCamp. https://www.datacamp.com/tutorial/how-to-exit-python-a-quick-tutorial

Python Software Foundation. (n.d.). *io — Core tools for working with streams*. Python Documentation. https://docs.python.org/3/library/io.html

Python Software Foundation. (2025). *uuid — UUID objects according to RFC 4122*. Python 3.13.3 documentation. https://docs.python.org/3/library/uuid.html

Python Software Foundation. (n.d.). *shutil — High-level file operations.* Python Documentation. https://docs.python.org/3/library/shutil.html#shutil.copy

Python Software Foundation. (n.d.). *pathlib — Object-oriented filesystem paths*. Python 3.13.3 documentation. https://docs.python.org/3/library/pathlib.html

Reitz, K., Benfield, C., Cordasco, I. S., & Prewitt, N. (2023). *Requests: HTTP for Humans* (Version 2.32.3) [Computer software]. Python Software Foundation. https://docs.python-requests.org/en/master/

Scarlett, R. (2023). *Why Python keeps growing, explained.* GitHub. https://github.blog/developer-skills/programming-languages-and-frameworks/why-python-keeps-growing-explained/

Sinyaniso, S. (2025). *What is SQLite — Understanding the Lightweight, Serverless Database.* Medium. https://sibabalwesinyaniso.medium.com/what-is-sqlite-understanding-the-lightweight-serverless-database-1b786b5d5e6e

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

WP Engine. (2023). *Why you Should Start Using Font Awesome.* WP Engine. https://wpengine.com.au/resources/why-you-should-start-using-font-awesome/