

# **Web Interface for classification of Photographic Images of Breeding Tropical Seabirds**

**Yuvraj Gavhane**

Registration number 100425274  
FINAL YEAR PROJECT - 2024

Supervised by Dr Michal Mackiewicz

# Abstract

This project's main objective is to provide a web interface for deep-learning object detection models with the particular purpose of categorizing Round Island petrels in time-lapse photos captured by trap cameras so that breeding trends can be seen. The project incorporates the pre-trained Faster R-CNN object detection model to interpret classification results and analyze breeding behaviors of Round Island petrels using React.js for the client side and FastAPI for the server side. The interface allows users to upload raw time-lapse camera photos, which are subsequently processed using the trained model to produce statistics on petrel observations. These figures provide important information about the habits and trends of Round Island petrel breeding. This study shows that by automating the otherwise labor-intensive and time-consuming process of time-lapse photography analysis, the use of an object classification web interface greatly improves seabird observation [Bieliajevaite, 2023].

# Acknowledgements

I would like to thank everyone who has contributed to the completion of this project.

# Executive Summary

The Wing Vision project was developed under the guidance of Professor Michal Mackiewicz, to create a user-friendly graphical interface that integrates a pre-trained model for non-specialist staff who lack computer science and coding expertise. The model was trained on images from a time-lapse camera project on Round Island, Mauritius, which has been monitoring the breeding patterns of petrels since December 2019 [Bieliajevaite, 2023].

The project aimed to overcome the challenge of using a model by non-technical staff, which was previously difficult and costly due to the need for human intervention. To address this, automated image analysis software was developed to perform detection and recognition tasks, reducing the cost and time required for analysis. The Wing Vision application allows users to detect petrels, chicks, and eggs from a predefined set of species without interacting with the underlying code.

The web-based application supports a computer vision framework that detects and recognizes adult, chick, and egg categories, drawing a box around each detection with a corresponding name. Users can upload a sequence of images, store the detected images in a database, and retrieve them based on the image creation date. The application utilizes High-Performance Computing (HPC) to process computationally intensive tasks, making image processing faster.

The Wing Vision project has successfully created a user-friendly interface that enables non-specialist staff to use a pre-trained model for image analysis, reducing the need for human intervention and costly technician services. The application's ability to process images quickly and efficiently makes it a valuable tool for researchers, conservationists, and wildlife photographers, among others, who can now focus on their work without being hindered by technical limitations.

# Abbreviations

Abbreviation	Definition
WVWA	Wing Vision Web Application
Yolo	You Only Look Once
GPU	Graphics Processing Unit
GUI	Graphical User Interface
OS	Operating System
HPC	High Performance Computer
UEA	University of East Anglia
SLURM	Simple Linux Utility for Resource Management

# List of Figures

2.1	Moscow analysis for web application . . . . .	4
3.1	System Architecture . . . . .	11
3.2	Login architecture . . . . .	12
3.3	Processing inside HPC . . . . .	13
3.4	Database tables . . . . .	14
3.5	Upload tab . . . . .	16
3.6	Download tab . . . . .	17
3.7	Images Selected . . . . .	17
3.8	Feedback mechanism: Running in HPC . . . . .	17
3.9	Validating SQL Database . . . . .	18
3.10	Job queue status . . . . .	18
3.11	Backend Fastapi API response log . . . . .	18
3.12	HPC Output log . . . . .	19
3.13	Downloaded images example . . . . .	19
3.14	HPC Job-status time (in minutes) . . . . .	20
3.15	Downloaded images . . . . .	20

# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>ii</b>
<b>Executive Summary</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Preparation and Design</b>	<b>3</b>
2.1 Login and Logout . . . . .	4
2.2 Main pages . . . . .	6
2.3 Models . . . . .	7
2.3.1 Faster R-CNN . . . . .	7
2.3.2 YOLO v7 . . . . .	8
2.4 Technologies . . . . .	8
2.4.1 React JS . . . . .	8
2.4.2 Fast API . . . . .	8
2.4.3 PostgreSQL . . . . .	9
2.4.4 Linux for HPC . . . . .	9
<b>3 Implementation and Evaluation</b>	<b>10</b>
3.1 System Architecture and Design . . . . .	10
3.2 Front end Implementation . . . . .	11
3.3 Back end Implementation . . . . .	12
3.4 HPC Integration . . . . .	13
3.5 Database Management . . . . .	14
3.6 Evaluation . . . . .	15
3.7 User testing and Frontend Validation . . . . .	16
<b>4 Conclusion</b>	<b>21</b>

# Chapter 1

## Introduction

New developments in computer vision have opened up creative applications in a range of fields, including security, healthcare, and wildlife monitoring. Sophisticated computer vision technologies specialized for time-lapse and video photography have been actively developed by researchers at the School of Computing Science (UEA). These tools, which are at the edge of technical innovation and offer reliable solutions for challenging visual data analysis tasks, were mostly created using machine learning frameworks like PyTorch.

Even though these computer vision algorithms are very strong, only those with a good knowledge of computer science and programming can use them in practice. Significant obstacles prevent these algorithms from being widely used, including the difficulty of setting up and operating them and the requirement for laborious pre- and post-processing procedures. This difficulty highlights the need for a more approachable user interface that can democratize access to these sophisticated instruments, allowing researchers, environmentalists, and other prospective users with little experience with programming to take advantage of computer vision's advantages.

By incorporating a sophisticated object detection model into a web-based interface, this initiative seeks to close this gap. The main goal is to create an intuitive platform that makes it easier to execute the model, increasing its applicability to a wider range of users. To provide end users with a seamless experience, the web interface will be built to handle the complete workflow, from inputting raw visual pictures acquired from various cameras to generating and visualizing the inference offered by the model.

This project is primarily concerned with web development, making use of contemporary web technologies to produce a user interface that is both simple to use and effective. The application's client-side will be created with React.js, a well-liked JavaScript toolkit that is excellent at creating interactive user interfaces quickly and easily [Blog, 2022]. FastAPI will be used on the server side to handle the backend tasks, such as processing uploaded images, forwarding them to HPC for object detection algorithms to be run there, returning the results to the user, and storing both the result and the raw image in a PostgreSQL



database for that specific user to maintain integrity and profile management [DataCamp, 2022, Team, 2022b, University of East Anglia and Services, 2022].

This project has great potential to improve the effectiveness and accessibility of wildlife monitoring, especially for research on seabird breeding habits like that of Round Island petrels. The web interface facilitates more accurate and thorough data analysis while also speeding up the study process by automating the labor-intensive activity of time-lapse photography analysis. In turn, this offers insightful information about the behaviors and breeding patterns of these species, aiding in their management and conservation.

The methods and design of the WVWA web application, the computer vision strategy we've selected, the software specification, the datasets and tests, and further resources are all covered in the parts that follow.

## Chapter 2

# Preparation and Design

The overall objective of this project was to develop an application that provides a user-friendly interface that allows Wildlife researchers, biologists, and Environmental Conservationists who do not have machine learning or programming expertise to use pre-trained computer vision and machine learning models for analyzing images, detecting and recognizing the adult, chick, and eggs of the petrels.

The developed system was built taking inspiration and incorporated into a unique design from the work done in the AVIMS project developed by [M. Mackiewicz and Fisher, 2023] and [Bieliajevaite, 2023]. Their work demonstrated the power of a Faster Region-based convolutional neural network over other more traditional approaches and suggested a possible system architecture. Recommendations by prof. Michal was followed closely, making several important refinements based on experience developing larger-scale systems capable of handling many more entities of interest.

Throughout the project, several meetings took place which helped to design and iteratively refine our approach so that the end product met requirements. These meetings with the professor included many architectures drawn, HPC account concerns were raised, and the professor provided the pre-trained YOLO v7 model trained by [Jenkins, 2023] as well as the code to interact with HPC. For testing the model via application, the dataset was provided.

With the help of our application, users without any prior knowledge of computer science or coding can execute computer vision models without having to deal with the underlying code. The program is compatible with computer vision models that share the same structure of identifying objects from a predetermined list of classes and creating an annotation bounding box labeled with the class name.

The workflow of our web-based application which implements the above computer vision framework is as follows:

<b>Must Have</b> <ol style="list-style-type: none"> <li>1. The web platform must be intuitive, enabling users with minimal technical background to easily navigate and utilize the detection system.</li> <li>2. The server-side implementation must efficiently handle image processing tasks, forward data to HPC for analysis, and return results promptly.</li> <li>3. All processed and raw images, as well as the analysis results, must be stored securely in a PostgreSQL database, ensuring data integrity and facilitating profile management.</li> </ol>	<b>Should Have</b> <ol style="list-style-type: none"> <li>1. The system should be designed to handle an increasing number of users and larger datasets as the user base grows.</li> <li>2. Comprehensive user guides and technical documentation should be available, supporting users in understanding the platform and its capabilities.</li> <li>3. The platform could offer real-time analysis and monitoring features, allowing users to observe ongoing data collection and analysis processes.</li> </ol>
<b>Could Have</b> <ol style="list-style-type: none"> <li>1. The platform could have provided options to add images and further retrain the model again with the added images, allowing for continuous improvement and adaptation of the model to new data</li> </ol>	<b>Won't Have</b> <ol style="list-style-type: none"> <li>1. The platform will not focus on providing image editing capabilities.</li> <li>2. There will be no options for basic pre-processing as well for analysis</li> </ol>

Figure 2.1: Moscow analysis for web application

The project priorities are outlined in the Moscow analysis, with the creation of a user-friendly online interface being emphasized as a must-have to enable non-experts to utilize the sophisticated object detection YOLO V7 model. One of the platform's primary features is the smooth backend processing of user-uploaded data, which is transferred to HPC (High-Performance Computing) computers for extensive processing before being returned to the platform for user access. This guarantees that complicated processing is performed effectively and doesn't put a strain on the user's local resources. To provide profile management and preserve data integrity, PostgreSQL requires strong data storage. Scalability and comprehensive documentation are noted as must-haves, enabling the platform to grow with the number of users and guaranteeing its continued availability to a wide range of users. To keep things simple, complex customization and picture editing features are not included because they are won't have.

## 2.1 Login and Logout

This web application's authentication and login system is built with React.js, with a focus on context management via a unique authentication context. The login interface is essential since it provides users with a safe and easy way to

access the program. React's hooks make efficient use of state management by enabling real-time modifications to form fields, error messages, and password visibility settings. Users are guaranteed to interact with a dynamic interface that offers instantaneous feedback thanks to this responsive design. The program connects with the backend to authenticate the user when the user enters their credentials. If the authentication token is successful, the program takes the user to the homepage and saves it. Because only authenticated users may access the application's protected regions, this system is based on the OAuth2 password bearer method, which manages token production and validation. To execute user authentication, a database query is run to retrieve stored credentials, which are then validated against the user's input. The system creates a JSON Web Token (JWT) after successful validation. This token is encrypted securely with a secret key and the HS256 [Reynanth, 2020] method. Following its return to the homepage, this token is saved and utilized to control the user's session, offering a safe way to preserve authentication throughout the application. An error notice is shown to the user right away if the credentials are incorrect, which improves user experience by making any problems obvious. In addition to managing login, the backend code includes a mechanism for protecting sensitive routes within the application. When a user attempts to access these protected areas, the system checks the validity of the provided JWT by decoding it and verifying the user's identity. If the token is valid and the user's credentials are correctly encoded within it, access is granted. Otherwise, the system prevents access by raising an appropriate error, ensuring that unauthorized users cannot reach restricted content.

The Login page is purposefully designed to be user-friendly, with careful consideration paid to both the functionality and aesthetics of the interface. A password visibility option that enables users to quickly transition between masked and unmasked password input has been added to further improve usability. This feature is essential for enhancing the user experience, particularly in settings where typos could happen. To further ensure security and administrative control over account management, the login form also offers assistance to users who may have forgotten their passwords, directing them to get in touch with the administrator.

The application has a thorough method for logging out in addition to signing in. Every trace of the user's session is carefully eliminated by this process, which also eliminates any saved authentication tokens and user data from local storage. It also makes sure that all session-related cookies are removed, which improves security even more by thwarting any unauthorized access. The user is taken to the login page when the session has been completely deleted, where they must authenticate anew before they may use the application once more. By carefully managing the logout process, you can preserve the security and integrity of the user's session and make sure that no private information is left exposed.

The AuthContext is the cornerstone of the application's authentication procedure. It provides a global context for managing the status of user authentication, including the saving and retrieval of tokens and usernames. Using React's context API, the AuthProvider component wraps the authentication logic and

makes it available to the entire application. This enables easy access to and modification of the authentication state by any app component. The Auth-Context login function ensures that tokens and users remain persistent between page refreshes and session lengths by storing them in local storage. On the other hand, to prevent unauthorized access and maintain the integrity of user sessions, the logout function ensures that all authentication data is entirely erased when a user logs out.

## 2.2 Main pages

The web application was created with an interactive interface with three main pages: an upload page for file management, a homepage showcasing the application's capabilities, and an about page for contacting the administrator. The application's design places a strong emphasis on a modular and understandable structure, especially in the file management section. This part is in charge of managing file uploading, processing, and downloading while preserving a flawless user interface. React hooks are used in the application to manage states and provide global data access, making it reactive and context-aware.

The tab-based architecture that organizes the user interface makes it simple for users to switch between various functionalities. Users may select, preview, and upload several files at once thanks to the simplified file management system. The interface makes it simple to remove files from the selection and offers visual feedback in the form of thumbnail previews of the files that are currently selected. The procedure used to manage file uploads to the backend makes sure that the files are handled effectively and formatted correctly. After uploading the files at the backend, a button is enabled, on clicking a button triggers a script that starts processing files; it updates users while processing is on and lets them see the processed files before downloading them.

The script automates the process of transferring files between a local machine and a remote server (HPC) using Paramiko library [Team, 2022c], executing commands on the server using Fabric library [Team, 2022a], and managing data after the remote processing is complete. Initially, it sets up a secure connection to the remote server using SSH, authenticated by a private key. Once connected, the script uploads specified files from the local machine to a designated directory on the remote server. After the files are uploaded, it remotely runs a specific shell script on the server, using a job scheduler [University of East Anglia and Services, 2022].

The shell script specifies to activate the specified environment and execute the pipeline file on the images stored on the HPC. Whilst the pipeline is executing, the scripts monitor the status of this job by repeatedly querying the job scheduler, ensuring that the job has been completed before proceeding further. This Python pipeline combines object detection, metadata extraction, and result visualization to automate image analysis. The first step is to search a directory for JPEG images. Then, using ExifTool, it extracts metadata such as "Date/Time Original" and saves it in a JSON file [GeeksforGeeks, 2020]. Next,

a YOLOv7 model is utilized by the pipeline to identify items in the photos, including "Adult," "Chick," and "Egg." The results are then stored in YOLO format and subsequently converted to the more commonly used COCO format. Bounding box dimensions are adjusted during the conversion process, and segmentation polygons are created and combined with the metadata. The pipeline then saves the annotated photos for visual verification, labels the discovered objects with their object category, and creates scaled bounding boxes around them. These JSON files containing metadata and the processed images are pulled from the HPC to the local storage for the user to download immediately if they wish to and it does store these images in the database as well under their own account.

The customizable architecture of the download functionality lets users receive processed files by giving parameters like date range and camera name. To make this procedure easier, the UI has elements like text inputs and date pickers. The program also provides a handy way to download all files that have been processed into a single package. The program pays close attention to user feedback and error handling throughout, making sure that users are aware of the consequences of their choices and that any problems are addressed clearly and understandably. Because of its careful design, the application manages the file lifecycle efficiently and is resilient, responsive, and easy to use.

## 2.3 Models

It was first suggested that the F-RCNN model, trained on the Petrel dataset by [Bieliajevaite, 2023], be used for this project. Moreover, Yolo v7 was developed and trained by [Jenkins, 2023] to be used, as it is a lightweight program. Though F-RCNN has advantages of its own, Yolo, the next version, is said to work the best and be the lightest variant. More about these two particular models will be covered below.

### 2.3.1 Faster R-CNN

A novel object identification system called Faster R-CNN combines convolutional neural networks (CNN) with region proposal networks (RPN) to provide end-to-end object detection. From the feature maps, the architecture immediately produces region recommendations, which the network then classifies and refines. This strategy improves accuracy while drastically lowering the computing burden associated with conventional detection techniques. Faster R-CNN provided a scalable and effective way to identify things in images, which set the groundwork for many contemporary object identification models. [Ren et al., 2016]

### 2.3.2 YOLO v7

Significant improvements in real-time object detection are introduced by YOLO v7, which prioritizes accuracy and speed. To balance the network depth, width, and resolution, the architecture includes elements like compound scaling, model scaling techniques, and extended efficient layer aggregation networks (E-ELAN). Compared to earlier iterations and other modern models, these enhancements allow YOLOv7 to attain state-of-the-art performance with faster inference times and higher accuracy. YOLOv7 is especially effective on edge devices, which makes it perfect for activities demanding fast and accurate object detection. [Wang et al., 2022]

We can see that YOLOv7 performs faster and more accurately than Faster R-CNN, particularly in real-time applications. The idea of region proposal networks was first presented by Faster R-CNN to increase detection efficiency. However, YOLOv7 goes one step further by predicting bounding boxes and class probabilities in a single network run, doing away with the requirement for a separate proposal phase. Because of its design, YOLOv7 can function at much faster frame rates while maintaining a high level of detection accuracy, which makes it more appropriate for real-time and resource-constrained settings.

## 2.4 Technologies

### 2.4.1 React JS

A robust JavaScript toolkit called React.js is used to create user interfaces, especially for single-page apps where responsive and dynamic interaction is essential. React.js was selected for client-side development in this project because of its adaptability and component-based architecture, which let programmers create reusable user interface components. This is necessary to provide a smooth and interesting user experience. React's virtual DOM makes it possible to update and render web pages quickly, which is crucial for applications that need to update in real-time and have a high level of interactivity, such as image processing or big dataset visualization. [?]

### 2.4.2 Fast API

This project will use FastAPI, a contemporary, high-performance web framework for creating Python APIs, for server-side development. FastAPI is based on the Starlette web server and has features like interactive API documentation, error management, and automatic data validation that simplify the process of developing web applications. FastAPI is renowned for its robust interaction with Python-based machine learning frameworks such as PyTorch, its automatic creation of interactive API documentation, and its ease of use. This makes it perfect for backend processing jobs including controlling data flow, responding to HTTP requests, and interacting with HPC equipment to run machine learning models. Because of its asynchronous features, FastAPI can effectively

handle several jobs at once, keeping the backend responsive even when there are high traffic volumes.[DataCamp, 2022]

### 2.4.3 PostgreSQL

Open-source and very capable, PostgreSQL is a relational database management system that supports complicated queries and is sturdy and extensible. PostgreSQL is utilized in this project to store photos and analysis results, offering a dependable and scalable method of handling massive amounts of data. Because of its sophisticated indexing methods and full-text search features, it's a great option for applications that need to get data quickly and perform intricate query operations. Furthermore, PostgreSQL is flexible in processing a range of data formats, which is advantageous for storing metadata with photos, thanks to its support for JSON and other non-relational data types.[Team, 2022b]

### 2.4.4 Linux for HPC

Bash is a scripting language used to manage jobs and automate job submissions in the HPC environment. Operating in UNIX-type settings, such as those found in the majority of HPC systems, requires the use of bash scripts. They make it possible to automate repetitive processes like sending jobs to schedulers (like SLURM) and managing file transfers between the local computer and the HPC system. Because of its ease of use, extensive application in high-performance computing (HPC) contexts, and capacity to carry out commands directly from the operating system's shell, Bash is the best option for overseeing the execution of computational operations in this project. [University of East Anglia and Services, 2022]



## Chapter 3

# Implementation and Evaluation

### 3.1 System Architecture and Design

The system was developed through a thorough design and implementation process that made use of contemporary backend and web technologies to produce a scalable, effective, and user-friendly platform. The system's architecture is based on a client-server paradigm, with FastAPI being used for the backend and React.js for the front end. PostgreSQL is used as the database management system, and high-performance computing (HPC) is integrated into the system for processing huge datasets.

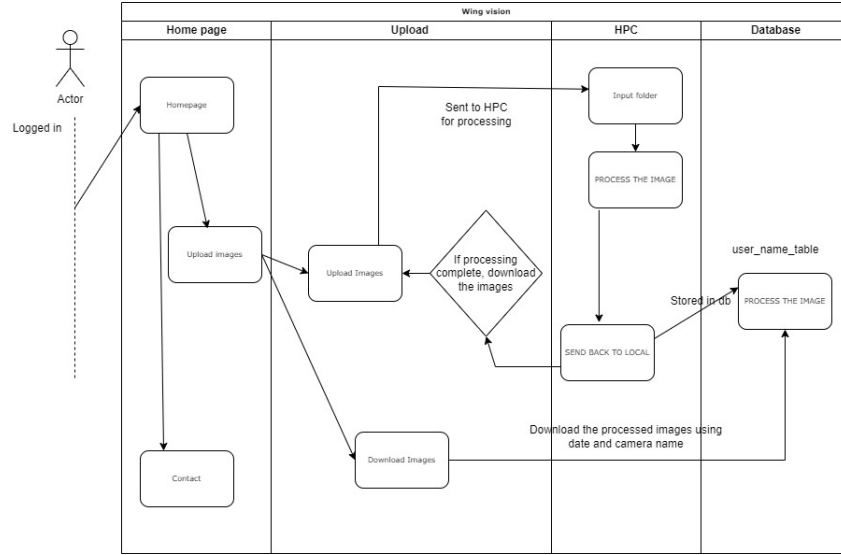


Figure 3.1: System Architecture

The architecture shown in the diagram represents a "Wing Vision" image processing system's workflow. The entire process of uploading, processing, and storing photographs is handled by the system. An actor, or user, who has logged in, interacts with the home page to upload photographs via the web interface to start the process. These photos are uploaded, and then they are processed extensively in a High-Performance Computing (HPC) environment. After processing the photos in an input folder, the HPC returns the processed images to the local system. The processed photos are downloaded back to the user's local environment if the processing is finished. Concurrently, the data processing is recorded in a database, linking the outcomes to tables specific to individual users. The architecture offers a reliable method for organizing and evaluating massive amounts of picture data by ensuring a smooth data flow between the user interface, the HPC for processing, and the database for storage.

## 3.2 Front end Implementation

React.js, a potent JavaScript package well-known for its capacity to create dynamic and interactive user interfaces, is used in the development of the system's client-side [Blog, 2022]. The home page, upload interface, and picture gallery are just a few of the components that correspond to the numerous capabilities of the system that make up the React application's structure. The main page acts as the application's hub, directing users through its main functionalities.

Through the upload interface, users may choose and upload photos straight from their devices. React.js manages user interactions and uses Axios to deliver

asynchronous HTTP requests to the backend, giving the user a non-blocking experience and real-time upload status updates. Maintaining a responsive user interface even with big files or sluggish network connections requires this interactivity [Watmore, 2020].

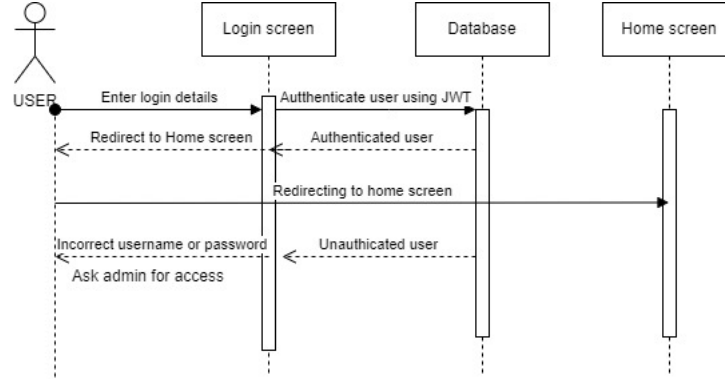


Figure 3.2: Login architecture

The user inputs their credentials on the login screen, which is subsequently confirmed by the backend. If authentication is successful, the user is redirected to the home screen; if not, an error prompt appears. This process is depicted in the login sequence diagram 3.1. Through JWT tokens, the system’s authentication flow guarantees safe application access, enabling a smooth user experience throughout sessions.

React’s component-based architecture also guarantees the front end’s scalability, maintainability, and modularity. Because every UI element—from buttons to forms—is contained within its component, it is simpler to control the behavior and state of the entire program. Because individual components may be changed or replaced without affecting the system as a whole, modularity also makes upgrades and maintenance easy [Blog, 2022]r.

### 3.3 Back end Implementation

FastAPI is a cutting-edge, scalable, and performance-optimized Python web framework that was used in the construction of the system’s backend. FastAPI was selected because of its capacity to manage high-throughput situations, which makes it perfect for applications that must swiftly and effectively handle a huge volume of requests.

The HTTP queries that the React frontend sends are handled by FastAPI. Upon receiving a picture from the user, FastAPI receives the image data, carries out preliminary validations (such as verifying file types and sizes), and then gets the data ready for the HPC system to handle. Because FastAPI supports

asynchronous programming, it may process numerous requests at once, cutting down on user wait times[DataCamp, 2022].

FastAPI's ability to automatically generate interactive API documentation using ReDoc and Swagger UI is one of its primary advantages. This functionality is very helpful for development and testing. A more effective development process results from developers being able to quickly examine, verify, and comprehend the data models utilized in the application's API endpoints.

### 3.4 HPC Integration

The High-Performance Computing (HPC) component oversees the system's computational heavy lifting. Users upload images, which are then transmitted to the HPC environment for intensive processing. Depending on the needs of the application, the processing includes activities like drawing bounding boxes using the detected coordinates and extracting metadata from the image.

Bash scripts are used to handle the automation of the task submission procedure. A Bash script that sends a job to the HPC's job scheduler, such as SLURM, is activated upon the upload of an image. The script oversees the job's whole lifecycle, from submission to status monitoring and result retrieval when processing is finished. Because of its strength and simplicity, Bash is especially well-suited for this kind of environment. Its command-line features let it communicate directly with the system's resources.

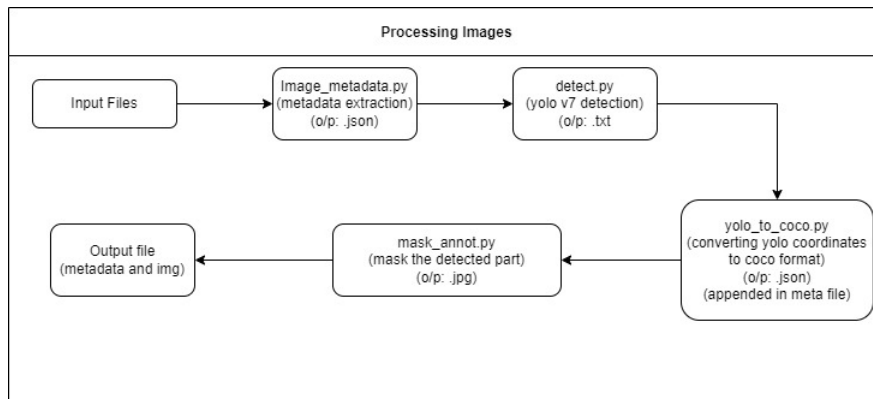


Figure 3.3: Processing inside HPC

The HPC environment's image processing workflow is depicted in the diagram. It creates a JSON file after first extracting metadata from the input photos. After that, these photos are examined with detect.py, which uses the YOLOv7 model to identify objects and generates a text file containing the coordinates of the detection. The data is translated to COCO format and appended to the JSON metadata for the coordinates. Lastly, bounding boxes are used

to mask the image, creating a new JPG file. The result of the procedure is an output that includes treated photos and improved metadata, both of which are available for additional usage. By using this method, the system is guaranteed to be able to manage processing activities that require a lot of resources and complexity without taxing the local server. Additionally, it makes use of the HPC's parallel processing capabilities, which enable the processing of numerous jobs at once and cut down on processing time.

### 3.5 Database Management

PostgreSQL is used as the database management system to store user data and process picture data permanently. PostgreSQL is selected for the rigorous data management requirements of "Wing Vision" because of its dependability, extensibility, and support for complicated queries.

The `username_images` and `login_details` tables are the two primary tables that the database structure is intended to effectively manage and organize data across. User credentials are stored in the `login_details` table, guaranteeing safe system access. Hashing methods are used to store passwords securely, and JSON Web Tokens (JWT) are used to manage user authentication. This ensures that sessions are stateless and safe.

All of the information about the photographs that users upload is managed by the `username_images` table. The binary data for the photos is stored in this table together with the image filenames and metadata. JSON-formatted metadata columns contain pertinent details including the date of upload, camera specs, and other essential information. Because PostgreSQL supports JSON, sophisticated queries that can extract particular metadata information straight from the database are made possible, allowing for flexible and effective querying.

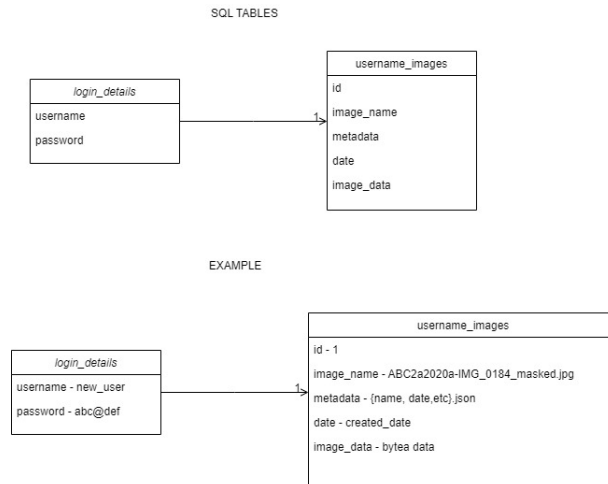


Figure 3.4: Database tables

The database's structure and the links between its tables are depicted in the Entity-Relationship Diagram (ERD), which is presented in Figure X. The `login_details` and `username_images` tables are specifically shown in this graphic, which also shows how user data and processed images are effectively and safely kept.

## 3.6 Evaluation

The WVWA system is being evaluated with an emphasis on evaluating its overall reliability, security, usability, and performance in real-world scenarios. This assessment aids in making sure the system achieves its goals and benefits its users. The effectiveness of "Wing Vision" is determined on how well it can handle and process massive amounts of picture data. React.js and FastAPI work together to guarantee that there is no performance decrease when handling a large number of concurrent requests on the front end and back end. FastAPI can handle several tasks at once because of its asynchronous features, which guarantee that consumers encounter the least amount of latency possible when interacting [DataCamp, 2022]. Performance is further improved by the interaction with the HPC, which transfers resource-intensive activities off the local server. Large dataset processing times can be greatly decreased by using the HPC's parallel processing capabilities, which allow the system to manage several image processing tasks at once. Because of its scalability, WVWA can expand to meet rising demand, which makes it appropriate for both small- and enterprise-scale applications.

The WVWA system's design and execution place a high priority on the user experience. The user-friendly interface of the React.js frontend facilitates the uploading of photos, the tracking of processing progress, and the downloading of outcomes. Users are guaranteed to be informed about the status of their requests at all times through the use of real-time feedback mechanisms like progress bars and status updates.

Users may easily access the system without compromising security thanks to the smooth login and authentication process. Because JWT is used for session management, users may stay logged in safely, which lessens the need for frequent re-authentication and enhances user experience in general.

Given the sensitive nature of user data and the possibility of unauthorized access, security is an essential component of the WVWA system. The system uses security techniques, such as encrypted communication channels between the frontend and backend, JWT for secure and stateless session management, and safe password storage via hashing methods. Furthermore, the database is built with security in mind, guaranteeing that user information is kept secure and that access is strictly regulated. In further iterations, role-based access controls could be added to further improve security by guaranteeing that only authorized individuals can access particular features or datasets.

The system's capacity to manage multiple users, big datasets, and intricate processing jobs without any issues is how reliable it is tested. Strong architecture

is the goal, and failures are gently handled using fail-safes. For instance, the system can attempt again or notify the user of the problem if an image processing job on the HPC fails. The system's ability to consistently store and retrieve massive volumes of data, even under extreme load, is ensured by the usage of PostgreSQL. To guard against data loss and guarantee that user data is secure even in the case of a system failure, backup and recovery procedures are also included.

### 3.7 User testing and Frontend Validation

Extensive user testing was carried out on both the frontend and backend components to replicate real-world usage and confirm that the system operates as intended under typical circumstances. A set of participants who represented the target population participated in user testing sessions. Testers were required to conduct operations including uploading photos, starting HPC processing, and downloading findings through the system's user interface. The system's responsive design, feedback mechanism, and easy navigation were praised by users in the overwhelmingly positive response. You can find attached screenshots of the user interface taken during testing that demonstrate successful downloads and uploads.

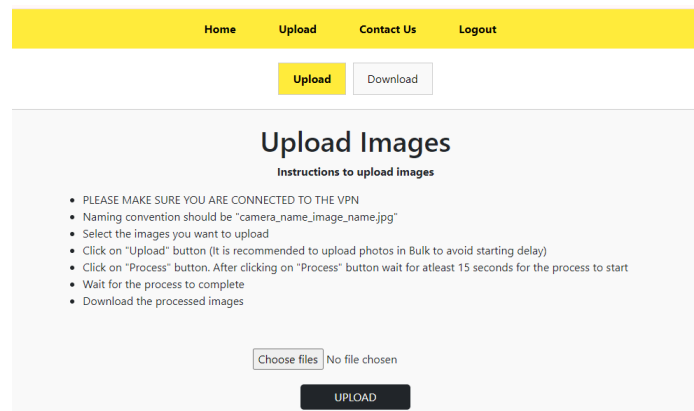


Figure 3.5: Upload tab

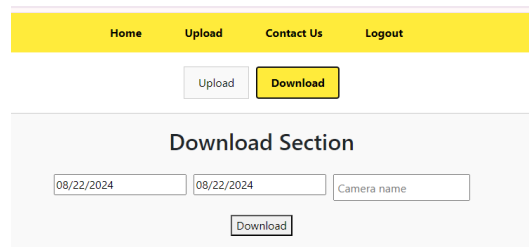


Figure 3.6: Download tab

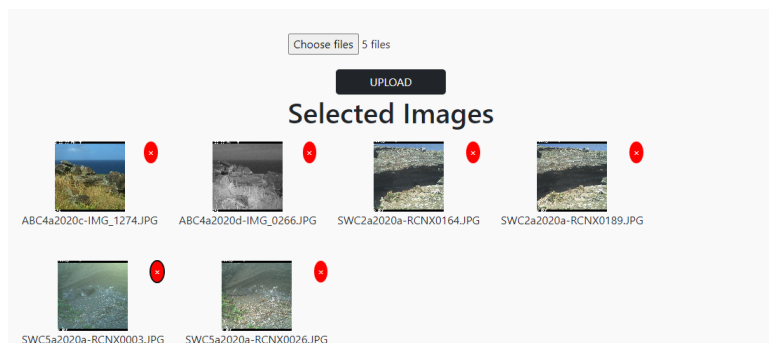


Figure 3.7: Images Selected

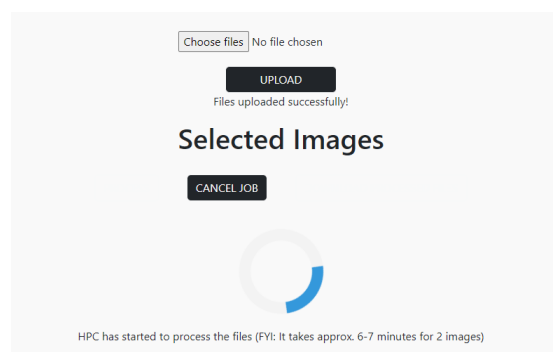


Figure 3.8: Feedback mechanism: Running in HPC

During these user testing sessions, the backend services—specifically, the FastAPI server and its interaction with the PostgreSQL database—were also verified. Users stated that the HPC performed image processing jobs accurately and that the API responded quickly. The findings were accessible and returned



in the anticipated time frames. During these sessions, the system's capacity to manage several simultaneous user requests was also tested, and no deterioration in performance was seen. You can find the attached screenshots of the processed results and API answers that the HPC returned, along with any logs from the user testing sessions that confirm the work was completed successfully.

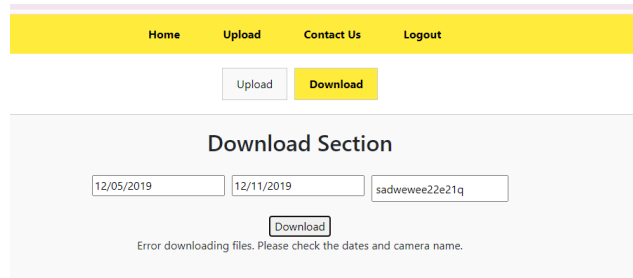


Figure 3.9: Validating SQL Database

```
[cdu23byu@login01 ~]$ squeue -u cdu23byu
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
18450004 compute-6 pipeline cdu23byu PD 0:00 1 (Priority)
[cdu23byu@login01 ~]$ squeue -u cdu23byu
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
18450004 compute-6 pipeline cdu23byu R 1:20 1 c0116
```

Figure 3.10: Job queue status

```
Files stored in download folder
INFO: 127.0.0.1:60710 - "GET /download?username=new_user&start_date=05-12-2019&end_date=11-12-2019&camera_name=ABC3a HTTP/1.1" 200 OK
Deleted file: C:\Old_HD_files\Wsc_DS_subject\DISSERTATION\cv_web_interface\wdng_vision_server\assets\download_images\ABC3a2020a-RC00020_bones.jpg
Deleted file: C:\Old_HD_files\Wsc_DS_subject\DISSERTATION\cv_web_interface\wdng_vision_server\assets\download_images\processed_stored_images.zip
All files have been deleted.
new_user 2019-12-05 2019-12-11 sadmewee22e21q
Query executed successfully
Files stored in download folder
INFO: 127.0.0.1:62292 - "GET /download?username=new_user&start_date=05-12-2019&end_date=11-12-2019&camera_name=sadmewee22e21q HTTP/1.1" 500 Internal Server Error
[]
```

Figure 3.11: Backend Fastapi API response log

```

Processing image: SMC5a2020a-RCIX00055.JPG
Metadata extracted: {'image_metadata': {'Date/Time Original': '2019:12:06 22:00:00'}}, File name: SMC5a2020a-RCIX00055
Running detection for SMC5a2020a-RCIX00055.JPG with command: python /gpfs/home/cdu23byu/vw_server_hpc/models/yolov7/detect.py --weights /gpfs/home/cdu23byu/vw_server_hpc
Detection completed for SMC5a2020a-RCIX00055.JPG.
YOLO annotations converted to COCO format and saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00055_metadata.json
Metadata with COCO annotations: {'annotations': [{'id': 1, 'category_id': 0, 'bbox': [448.0, 409.0, 580.0, 208.0], 'area': 120640.4, 'segmentation': [[448.0, 408.999398
Bbox image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00055_bboxes.jpg
Masked image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00055_bboxes.jpg
Processing image: ABC3a2020a-RCIX00002.JPG
Metadata extracted: {'image_metadata': {'Date/Time Original': '2019:12:04 18:00:00'}}, File name: ABC3a2020a-RCIX00002
Running detection for ABC3a2020a-RCIX00002.JPG with command: python /gpfs/home/cdu23byu/vw_server_hpc/models/yolov7/detect.py --weights /gpfs/home/cdu23byu/vw_server_hpc
Detection completed for ABC3a2020a-RCIX00002.JPG.
YOLO annotations converted to COCO format and saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00002_metadata.json
Metadata with COCO annotations: {'annotations': [{'id': 1, 'category_id': 0, 'bbox': [66.0, 238.5, 184.0, 170.0], 'area': 31280.01, 'segmentation': [[65.99987200000001,
Bbox image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00002_bboxes.jpg
Masked image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00002_bboxes.jpg
Processing image: ABC3a2020a-RCIX00026.JPG
Metadata extracted: {'image_metadata': {'Date/Time Original': '2019:12:05 18:00:00'}}, File name: ABC3a2020a-RCIX00026
Running detection for ABC3a2020a-RCIX00026.JPG with command: python /gpfs/home/cdu23byu/vw_server_hpc/models/yolov7/detect.py --weights /gpfs/home/cdu23byu/vw_server_hpc
Detection completed for ABC3a2020a-RCIX00026.JPG.
YOLO annotations converted to COCO format and saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00026_metadata.json
Metadata with COCO annotations: {'annotations': [{'id': 1, 'category_id': 0, 'bbox': [-17.0, 261.0, 372.0, 160.0], 'area': 59519.95, 'segmentation': [[-17.00003840000000
Bbox image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00026_bboxes.jpg
Masked image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/ABC3a2020a-RCIX00026_bboxes.jpg
Processing image: SMC5a2020a-RCIX00003.JPG
Metadata extracted: {'image_metadata': {'Date/Time Original': '2019:12:04 18:00:00'}}, File name: SMC5a2020a-RCIX00003
Running detection for SMC5a2020a-RCIX00003.JPG with command: python /gpfs/home/cdu23byu/vw_server_hpc/models/yolov7/detect.py --weights /gpfs/home/cdu23byu/vw_server_hpc
Detection completed for SMC5a2020a-RCIX00003.JPG.
YOLO annotations converted to COCO format and saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00003_metadata.json
Metadata with COCO annotations: {'annotations': [{'id': 1, 'category_id': 0, 'bbox': [-12.5, 404.5, 710.0, 290.0], 'area': 205900.09, 'segmentation': [[-12.50099199999999
Bbox image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00003_bboxes.jpg
Masked image saved to /gpfs/home/cdu23byu/vw_server_hpc/assets/output/SMC5a2020a-RCIX00003_bboxes.jpg
Total time taken: 403.4410481452942
Job completed on Thu 22 Aug 12:58:11 BST 2024

```

Figure 3.12: HPC Output log

User comments led to a positive evaluation of the system’s overall performance. The fact that users were able to do every task without running into serious problems shows that the system is stable. The data transmission between the local environment and the HPC was effective, and the processing times for picture jobs were within reasonable bounds. The findings of user testing indicate that the system is sufficiently resilient to withstand the anticipated workload during its initial deployment phase, even though more testing—such as load or stress testing—would be advantageous. Screenshots of performance metrics, such as average processing times and successful task completion images in a zip folder, are included.

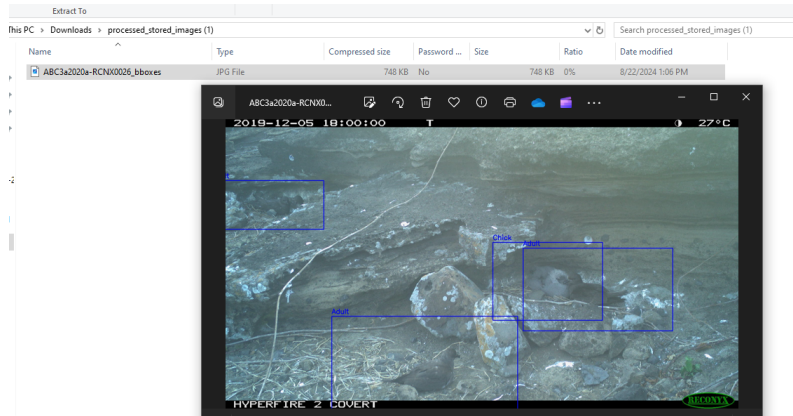


Figure 3.13: Downloaded images example

```
Job is still running. Waiting...
Job status:
JOBID PARTITION NAME USER ST TIME NODES MODEL1ST(REASON)
18460008 compute-6 pipeline cd023byu R 6:57 1 c0119
```

Figure 3.14: HPC Job-status time (in minutes)

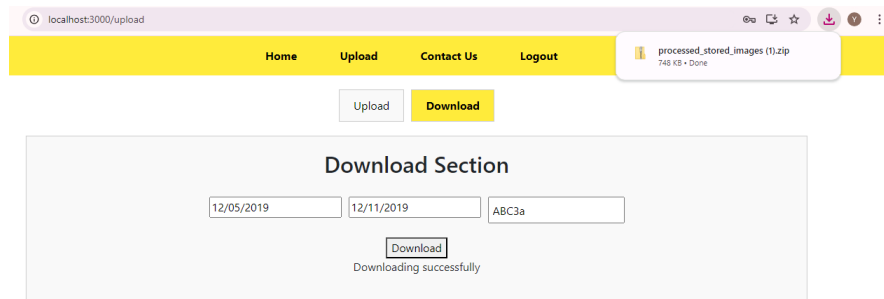


Figure 3.15: Downloaded images

## Chapter 4

# Conclusion

The results of the performance and user testing provided insightful information about the general effectiveness of the system as well as possible areas for development. The time needed for image processing was greatly reduced because to the integration with High-Performance Computing (HPC), which proved to be quite efficient. This feature, which enables the system to efficiently tackle computationally demanding tasks, is essential to its core functionality. On the other hand, a bottleneck was indicated by a delay in the start of processing jobs, which was probably caused by the time needed to activate the HPC environment. While not life-threatening, this delay indicates that the activation mechanism of the environment can be optimized to improve performance even further. Users have given the user interface, which was created with an emphasis on intuitiveness, and positive reviews, highlighting how user-friendly it is. On the other hand, some users proposed enhancements including incorporating a way to re-train the model with more photos and faster loading speeds for larger datasets. To better satisfy user expectations and requirements, these proposals offer a clear direction for future versions of the system.

Even though the system performed admirably, the investigation revealed a few crucial areas that need more work. Because the system depends so heavily on the speed and availability of HPC resources, there is a risk that these resources could become scarce or unavailable. On the one hand, HPC speeds up processing, but there is also a risk that they could become unavailable. Future studies could investigate integrating cloud-based processing options to address this issue. This would lessen reliance on a single HPC resource and possibly provide more flexible and scalable processing capabilities. And there's still more room for improvement: database query optimization. Simplified queries have the potential to greatly improve system performance during periods of high workload, guaranteeing that the system stays responsive and effective even in the face of challenging circumstances.

All things considered, the system proved to be very reliable, handling numerous users at once and massive amounts of data without experiencing any serious problems. The PostgreSQL database turned out to be a reliable option

for controlling the storage requirements of the program, efficiently handling the sizable datasets that the system created and processed. In addition, the system's overall security was enhanced by the efficient and safe way in which user sessions were maintained through the use of JSON Web Tokens (JWT) for user authentication. Testing was also done to validate the integration with the HPC, demonstrating that the system can reliably execute and finish image processing operations even in high-demand settings. Even if the system as it is now operates well and is dependable, these results point to specific directions for future development, guaranteeing that the system can adapt to changing user needs and keep up with current trends.

# Bibliography

- [Bieliajevaite, 2023] Bieliajevaite, A. (2023). Classification of photographic images of breeding tropical seabirds. Technical report, Final Year UG Project - School of Computing Sciences, University of East Anglia, 2023.
- [Blog, 2022] Blog, H. (2022). React.js: A beginner’s guide to building fast, scalable uis. <https://blog.hubspot.com/website/react-js>.
- [DataCamp, 2022] DataCamp (2022). Introduction to fastapi tutorial. <https://www.datacamp.com/tutorial/introduction-fastapi-tutorial>.
- [GeeksforGeeks, 2020] GeeksforGeeks (2020). Installing and using exiftool on linux. [Online; accessed ;date;].
- [Jenkins, 2023] Jenkins, M. (2023). Detecting tropical birds in time-lapse trap camera imagery. Technical report, Final Year UG Project - School of Computing Sciences, University of East Anglia, 2023.
- [M. Mackiewicz and Fisher, 2023] M. Mackiewicz, G. F. and Fisher, M. (2023). A new application for automated video identification of marine species (avims). Accessed on 2024-04-23.
- [Ren et al., 2016] Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks.
- [Reynanth, 2020] Reynanth, B. V. S. (2020). Comparison of rs256 and hs256 algorithms for token signing in cryptography. *Medium*. <https://medium.com/@bvsreynanth/comparison-of-rs256-and-hs256-algorithms-for-token-signing-in-cryptography-bd21e9e7a54d>.
- [Team, 2022a] Team, F. (2022a). Fabfile: A python library for automating tasks. <https://www.fabfile.org/>.
- [Team, 2022b] Team, P. (2022b). About postgresql. <https://www.postgresql.org/about/>.
- [Team, 2022c] Team, P. (2022c). Paramiko: A python implementation of sshv2 and sftp. <https://www.paramiko.org/>.

- [University of East Anglia and Services, 2022] University of East Anglia, I. and Services, C. (2022). High-performance computing (hpc) service. <https://my.uea.ac.uk/divisions/it-and-computing-services/service-catalogue/research-it-services/hpc>.
- [Wang et al., 2022] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.
- [Watmore, 2020] Watmore, J. (2020). React axios http post request examples. [Online; accessed 18/08/2024].