**Pathway:**XBCAD7329

**Group Name:**TymeMA

**Group Members:**Ted Ngobeni - ST10027949  
Tshiamo Thekiso – ST10132516  
Motjoka Fanana – ST10089515

**Scrum Board URL:**<https://teamfu.tech/dashboard/4d73d4ec-7fc0-41e8-92a1-3a5693f2f160/settings>

**Github**:   
<https://github.com/TymeMAORG/LoryParkZoo>

**Client website**:  
<https://www.lorypark.co.za/>

Contents

[Introduction 4](#_Toc180446965)

[Research into the Client Organization 4](#_Toc180446966)

[Work Agreement 5](#_Toc180446967)

[Ethical and Privacy Concerns 7](#_Toc180446968)

[Definition of Ready (DoR) 7](#_Toc180446969)

[Definition of Done (DoD) 8](#_Toc180446970)

[Requirements 9](#_Toc180446971)

[User Roles 9](#_Toc180446972)

[User Stories 9](#_Toc180446973)

[Product Backlog 12](#_Toc180446974)

[Sprint Planning: 16](#_Toc180446975)

[User Experience (UX) mapping 18](#_Toc180446976)

[Admin User Experience One 18](#_Toc180446977)

[Admin User Experience Two 19](#_Toc180446978)

[Admin User Experience Three 20](#_Toc180446979)

[Staff User Experience One 20](#_Toc180446980)

[Staff User Experience Two 21](#_Toc180446981)

[Staff User Experience Three 22](#_Toc180446982)

[Staff User Experience 23](#_Toc180446983)

[Staff User Experience Four 24](#_Toc180446984)

[Staff User Experience Five 24](#_Toc180446985)

[Non-functional Requirements 25](#_Toc180446986)

[Analysis artifacts 29](#_Toc180446987)

[Implementation Documentation 30](#_Toc180446988)

[Data Scheme Documentation 31](#_Toc180446989)

[Architecture artifacts 32](#_Toc180446990)

[Security 33](#_Toc180446991)

[DevOps 34](#_Toc180446992)

[GitHub Actions Pipeline overview 34](#_Toc180446993)

[Pipeline Workflow Diagram 36](#_Toc180446994)

[Running costs 37](#_Toc180446995)

[Change Management 39](#_Toc180446996)

[Appendices 40](#_Toc180446997)

# Introduction

The Internal Zoo Staff Mobile App project is aimed at improving operational efficiency and data management at Lory Park Zoo. This app will assist the zoo’s staff in daily animal monitoring, enclosure maintenance, and reporting, while providing management with tools to oversee staff roles and responsibilities. The need for this app became evident during our team’s volunteer work at Lory Park Zoo, where we observed that staff rely heavily on manual, paper-based processes to track daily tasks such as animal feeding, reptile room maintenance, and environmental monitoring. These processes are prone to delays, inaccuracies, and data loss. The app will digitize these tasks, enabling streamlined data entry, real-time updates, and better administrative control.

During our volunteer experience, we recognized that each staff member specializes in specific animal groups and cannot always access other areas of the zoo. This understanding shaped the key feature of the app: restricted access based on staff roles, ensuring that only authorized personnel can manage their assigned sections. The app will also include essential daily and weekly forms for food monitoring, animal checklists, and environmental reports, making it an indispensable tool for both the staff and management.

## Research into the Client Organization

Our research into Lory Park Zoo began with our hands-on volunteer experience. We immersed ourselves in the daily routines of the zoo staff to better understand their workflow and the pain points in their current manual processes. From feeding schedules to enclosure checks, we observed that zoo employees juggle multiple tasks throughout the day, often relying on memory or physical logbooks to record important information. We also noted that environmental factors, such as weather conditions, affect the health and safety of the animals, and these details need to be accurately tracked and reported.

We held discussions with Robynn Ingle-Moller and Kyle from Lory Park Zoo to understand the administrative challenges they face, especially around keeping track of staff activities and the daily upkeep of animal enclosures. They expressed interest in a solution that not only improves record-keeping but also reduces the risk of human error.

This research reinforced the need for a custom mobile app that digitizes the zoo’s operations, providing more efficiency in data collection and reporting, as well as better oversight from management.

## Work Agreement

*Signed work agreement attached as Annexure 1*

Our team consists of three members with similar skillsets, and we have agreed on a collaborative and flexible approach to ensure balanced workload distribution. Since all members are equally proficient in areas such as frontend development, backend implementation, and testing, we decided to share responsibilities across the project while maintaining clear communication and accountability. The following work agreement was created and adhered to throughout the project:

1. **Shared Roles and Responsibilities**:
   * All members are responsible for both frontend and backend development tasks, ensuring everyone contributes to each part of the project.
   * Each member is involved in implementing key features such as the login system, CRUD operations for staff and animals, restricted section access, and form submissions.
   * Testing and quality assurance tasks are shared equally, with each member reviewing the other’s work to ensure robust functionality and error-free deployment.
2. **Communication**:
   * We scheduled **bi-weekly meetings** to review progress and discuss any challenges. Meetings were conducted on **Mondays and Thursdays** via Zoom.
   * For day-to-day communication, we used **Slack** for instant messaging and task discussions.
   * **Trello** was used for task management, where all members tracked the status of their assigned features and tasks.
3. **Workload Distribution**:
   * Each feature of the app (e.g., login system, daily forms, restricted section access) was divided into smaller subtasks, and members rotated between them. For example, one member would work on the backend of the login system, while another handled frontend design, and then they would switch roles for the next feature.
   * Any member encountering difficulties with a task was encouraged to seek support from the team, ensuring no one worked in isolation or under excessive pressure.
4. **Collaboration and Pair Programming**:
   * Pair programming sessions were held for complex features like restricted section access and role-based permissions. This allowed us to troubleshoot together and share ideas, leading to higher-quality code.
   * We also conducted **code reviews** at the end of each major development phase, providing feedback and helping each other improve the codebase.
5. **Decision-Making Process**:
   * Decisions regarding database architecture, security measures, and design choices were made **collectively**. In case of differing opinions, a vote was taken, and majority consensus determined the final choice.
   * During discussions, every member had the opportunity to present pros and cons of each approach, ensuring that decisions were informed and well-reasoned.
6. **Quality Control**:
   * Testing was integrated into our workflow. Each member was responsible for writing unit tests for their respective features, and peer testing ensured no features were deployed without review.
   * In addition, we rotated roles during testing phases, so no one tested their own work, ensuring objectivity and thorough bug detection.
7. **Ethical Responsibility and Privacy**:
   * All members agreed to uphold ethical standards when handling zoo data, such as restricting access to confidential information and protecting the privacy of staff members and animals.
   * Data encryption protocols and secure data storage were implemented by all members to prevent unauthorized access and protect sensitive data, particularly staff and animal details.
8. **Flexibility**:
   * Given the similarity in skillsets, the team-maintained flexibility to switch tasks as needed. If one member completed their part of a task early, they could assist others or take on tasks that were behind schedule.
   * This allowed for more adaptive project management, preventing bottlenecks and ensuring steady progress.

This work agreement ensured that all members contributed equally, leveraging our similar skillsets to collaboratively deliver a high-quality mobile app that meets the needs of Lory Park Zoo. The shared responsibility fostered teamwork and transparency, allowing us to complete the project efficiently and ethically

## Ethical and Privacy Concerns

The app will handle sensitive operational data about zoo animals and staff responsibilities. To maintain ethical standards, we are ensuring that:

1. **Data Privacy**: Only authorized users (e.g., zoo management) will have access to confidential information such as staff roles and responsibilities. Role-based access controls will be enforced to prevent unauthorized access.
2. **Data Security**: All data will be encrypted during storage and transfer to protect against unauthorized access and breaches.
3. **Ethical Reporting**: No animal-related data will be shared externally without explicit permission from the zoo management. Sensitive reports such as medical data or animal behaviour should remain confidential and accessible only to authorized personnel.
4. **Transparency**: The app’s terms of use will clearly outline how data is collected, used, and stored, ensuring transparency for all users.

## Definition of Ready (DoR)

Our team's **Definition of Ready** means that a task or feature is fully prepared and clearly defined before any development begins. The following criteria must be met for a task to be considered "ready":

* **Requirements are clear**: The task is well-documented with clear objectives and expected outcomes.
* **Acceptance criteria are defined**: We know exactly what will be tested and how the feature will function once complete.
* **Dependencies are resolved**: Any tools, APIs, or external resources needed to complete the task are available and working.
* **Team understanding**: All members have reviewed and understand the task, and we have no outstanding questions about what needs to be done.
* **Effort is estimated**: We have discussed and agreed on the estimated time or effort needed to complete the task.

## Definition of Done (DoD)

Our team's **Definition of Done** means that a task or feature is fully completed, meets quality standards, and is ready for deployment. The following criteria must be satisfied for a task to be considered "done":

* **Code is written**: The feature has been developed, and all relevant code is complete.
* **Testing is complete**: Unit tests and functional tests have been written and successfully passed.
* **Peer-reviewed**: Another team member has reviewed the code to ensure it meets quality and security standards.
* **Bug-free**: No critical bugs or issues are present, and any bugs found during testing have been fixed.
* **Documentation is updated**: All relevant documentation, including comments, user guides, and code documentation, has been updated.
* **Meets acceptance criteria**: The feature fulfills all predefined acceptance criteria and works as expected.
* **Feature is integrated**: The feature has been successfully integrated into the app and functions correctly in the broader system.

# Requirements

User Roles

1. **Admin (Management)**
   * Responsible for overseeing the zoo’s operations within the app.
   * Can manage staff, assign roles/sections, manage animal groups and camps, and perform CRUD operations on species, animals, and forms.
   * Has access to all sections and forms.
2. **Staff**
   * Assigned to specific animal groups/sections by the admin.
   * Can only view and complete forms related to their assigned section(s).
   * Responsible for daily and weekly form submissions, like food monitoring, checklists, and reports.

User Stories

**1. Admin (Management) User Stories**

* **As an admin, I want to create, update, and remove staff members, so that I can manage who has access to the app.**
  + **Acceptance Criteria**: Admin can add new staff members, update their assigned sections, or remove them from the system.
* **As an admin, I want to assign specific animal groups (sections) to each staff member, so that they can only access relevant sections of the zoo.**
  + **Acceptance Criteria**: Admin can assign one or more sections (e.g., big cats, reptiles) to each staff member.
* **As an admin, I want to add, update, and remove animal groups and camps, so that I can maintain accurate information about the zoo’s enclosures.**
  + **Acceptance Criteria**: Admin can manage animal groups (e.g., primates, exotic birds) and associated camps (enclosures).
* **As an admin, I want to add animals with their relevant information to animal groups, so that staff can view details about the animals they care for.**
  + **Acceptance Criteria**: Admin can input animal details (species, name, health info) and associate them with specific animal groups.
* **As an admin, I want to CRUD species, so that I can keep the animal records up to date and accurate.**
  + **Acceptance Criteria**: Admin can create, read, update, and delete species data in the app.
* **As an admin, I want to view and edit all forms submitted by staff, so that I can monitor zoo operations and ensure compliance.**
  + **Acceptance Criteria**: Admin can access and edit daily food monitoring sheets, checklists, and other reports.

**2. Staff User Stories**

* **As a staff member, I want to log in and be taken directly to my assigned animal section, so that I can quickly access the relevant part of the app.**
  + **Acceptance Criteria**: Upon login, the staff member is automatically redirected to their assigned section's dashboard.
* **As a staff member, I want to complete a daily food monitoring form for my section, so that I can record the animals’ feeding data.**
  + **Acceptance Criteria**: Staff can input food monitoring data, such as feeding times and portions, into the daily form.
* **As a staff member, I want to fill out a daily checklist for the animals in my section, so that I can report on their health and well-being.**
  + **Acceptance Criteria**: Staff can submit a checklist with observations like animal health, enclosure conditions, and behavior.
* **As a staff member, I want to complete the reptile room daily monitoring sheet, so that I can track the conditions in the reptile section.**
  + **Acceptance Criteria**: Staff working in the reptile room can fill out a daily monitoring form, including temperature, humidity, and reptile conditions.
* **As a staff member, I want to submit a weekly filter and pump checklist, so that I can ensure equipment maintenance is logged.**
  + **Acceptance Criteria**: Staff can fill out a weekly checklist on filter and pump status and any issues.
* **As a staff member, I want to record the electric fence readings daily, so that I can track the integrity of the zoo’s security measures.**
  + **Acceptance Criteria**: Staff can log electric fence readings in a form that is submitted daily.
* **As a staff member, I want to record the weather conditions for the day, so that I can note how external factors may impact the animals.**
  + **Acceptance Criteria**: Staff can input daily weather data, such as temperature, rainfall, and wind speed.
* **As a staff member, I want to submit a monthly fire extinguisher report, so that I can document the status of fire safety equipment.**
  + **Acceptance Criteria**: Staff can log the condition and maintenance status of fire extinguishers on a monthly basis.

**Admin-Only User Stories for Forms**

* **As an admin, I want to view all staff submissions across different sections, so that I can review the daily operations in one place.**
  + **Acceptance Criteria**: Admin has access to all forms and reports submitted by staff in all sections.
* **As an admin, I want to edit any submitted reports, so that I can correct or add information as needed.**
  + **Acceptance Criteria**: Admin can open, modify, and resubmit any forms completed by staff members.

**Summary of User Roles and User Stories**

* **Admin**: Oversees staff, animals, and operations. Has full access to manage sections, animals, species, and forms.
* **Staff**: Limited to the sections assigned to them, responsible for daily and weekly reporting on animal care and zoo operations.

This structure ensures role-based access control and smooth workflow for both management and staff in the zoo's daily operations.

Product Backlog

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **As a...** | **User Stories** | **Sprint** | **Estimation** | **Priority** | **Status** |
| 1 | Administrator | As an admin, I want to create, update, and remove staff members, so that I can manage who has access to the app. | 1 | 5 | Must | To be started |
| 2 | Administrator | As an admin, I want to assign specific animal groups (sections) to each staff member, so that they can only access relevant sections of the zoo. | 1 | 3 | Must | To be started |
| 3 | Administrator | As an admin, I want to add, update, and remove animal groups and camps, so that I can maintain accurate information about the zoo’s enclosures. | 1 | 8 | Must | To be started |
| 4 | Administrator | As an admin, I want to add animals with their relevant information to animal groups, so that staff can view details about the animals they care for. | 1 | 5 | Should | To be started |
| 5 | Administrator | As an admin, I want to view and edit all forms submitted by staff, so that I can monitor zoo operations and ensure compliance. | 3 | 8 | Should | To be started |
| 6 | Staff Member | As a staff member, I want to log in and be taken directly to my assigned animal section, so that I can quickly access the relevant part of the app. | 1 | 5 | Must | To be started |
| 7 | Staff Member | As a staff member, I want to complete a daily food monitoring form for my section, so that I can record the animals’ feeding data. | 2 | 5 | Must | To be started |
| 8 | Staff Member | As a staff member, I want to fill out a daily checklist for the animals in my section, so that I can report on their health and well-being. | 2 | 8 | Must | To be started |
| 9 | Staff Member | As a staff member, I want to complete the reptile room daily monitoring sheet, so that I can track the conditions in the reptile section. | 2 | 8 | Must | To be started |
| 10 | Staff Member | As a staff member, I want to submit a weekly filter and pump checklist, so that I can ensure equipment maintenance is logged. | 3 | 3 | Should | To be started |
| 11 | Staff Member | As a staff member, I want to record the electric fence readings daily, so that I can track the integrity of the zoo’s security measures. | 3 | 5 | Should | To be started |
| 12 | Staff Member | As a staff member, I want to record the weather conditions for the day, so that I can note how external factors may impact the animals. | 3 | 5 | Must | To be started |
| 13 | Staff Member | As a staff member, I want to submit a monthly fire extinguisher report, so that I can document the status of fire safety equipment. | 3 | 3 | Should | To be started |
| 14 | Administrator | As an admin, I want to view all staff submissions across different sections, so that I can review the daily operations in one place. | 3 | 5 | Should | To be started |
| 15 | Administrator | As an admin, I want to edit any submitted reports, so that I can correct or add information as needed. | 3 | 5 | Should | To be started |
| 16 | Staff Member | Monitor fire extinguisher status | 3 | 3 | Should | To be started |

A screenshot of a computer

Description automatically generated

Figure 1 - Highest priority backlog

Sprint Planning:

**Sprint 1: User Management & Access Control**

* **Goal**: Set up the basic user management features for admin and Employees.
* **Tasks**:
  1. Admin creates, updates and removes employees (ID 1, ID 9 from Backlog (Product Backlog) (User Roles)).
  2. The administrator assigns part of the employee (ID 2 (product accumulation)).
  3. Implement role-based access control to limit Employees to their assigned sections.
  4. The administrator manages animal groups, camps and species (ID 3, ID 4 (Product Backlog)).

**Sprint 2: Form Submissions for Daily Monitoring**

* **Goal**: Enable Employees to submit daily forms for animal monitoring and general operations.
* **Tasks**:

1. Implement daily employee food monitoring schedule (ID 7 (Product balance)).

2. Build an animal health checklist submission system (ID 8(Product Backlog)).

3. Set up daily monitoring of crawl space (ID 9 (Product Balance)).

4. Ensure form information is submitted within 5 seconds

**Sprint 3: Advanced Employees Reporting and Admin Oversight**

* **Goal**: Expand admin oversight with advanced reporting and Employees tools.
* **Tasks**:

1. Administrators can check, edit and monitor employees to submit (ID 14, ID 15 (product deferral)).

2. Allow submission of weekly filter and pump checklist (ID 10 (product balance)).

3. Daily electronic fence readings submitted by Employees (ID 11 (Product Balance)).

**Sprint 4: Security, Maintenance, and Final Adjustments**

* **Goal**: Finalize security and reliability features while ensuring the app is stable.
* **Tasks**:

1. Encrypt all data to ensure safety during transport and rest

2. Implemented offline mode for employee reporting, synchronization when the app is online

3. Create a user-friendly interface with easy-to-navigate mobile-optimized forms

4. Role-based security (RBAC) provides appropriate access control

# User Experience (UX) mapping

## Admin User Experience One

Managing (Create, Read, Update and Delete) Staff record as well as updating an assinged user role for the staff member.

|  |  |  |
| --- | --- | --- |
| Admin (Management)  **Scenario**: Management needs to create a new staff member profile and assign them to their relevant role.  **Expectations**:   * Ability to Create Read Update and Delete staff records * Ability to Add roles to staff profiles | | |
| Log In | Create Profile | Assign Roles |
| 1. Log In to the Admin role. 2. View all staff records and ensure the staff record is not already existing. | 1. Select profile creatio tab through the application User Interface 2. Fill required form detailing all relevant infromation about the Staff Member. 3. Save staff profile and get redirected to the main Admin UI. | 1. View all staff records 2. Select the newly created. 3. Select edit and change the assigned role to the required role. |

## Admin User Experience Two

Managing (Create, Read, Update and Delete) Animal groups and camps as well as updating the records.

|  |  |  |
| --- | --- | --- |
| Admin (Management)  **Scenario**: Management needs to create a new Animal group and camps  **Expectations**:   * Ability to Create Read Update and Delete animal groups and camps * Ability to Add animal groups and camps | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Admin role. 2. View all animal groups and camps to ensure required record is not already existing. | 1. Select Animal group creatio tab through the application User Interface 2. Fill required form detailing all relevant infromation about the animal group. 3. Save animal group and get redirected to the main Admin UI. | 1. View all animal groups & camps 2. Select the required record. 3. Select edit and change the infromation. |

## Admin User Experience Three

Admin/Management wants to edit a report/form that was submitted by a staff member or saved to the database.

|  |  |  |
| --- | --- | --- |
| Admin (Management)  **Scenario**: Management needs to update record/form  **Expectations**:   * Ability to Create Read Update and Delete staff records | | |
| Log In | Selecting report/form | ----------- |
| 1. Log In to the Admin role. 2. View all staff records and ensure the staff record is there. | 1. Select report/form the admin wishes to update/edit 2. Fill required form detailing all relevant infromation required. 3. Save staff report/form and get redirected to the main Admin UI. | ----------- |

## Staff User Experience One

Staff member filling out a daily checklist form for animals that can be found in their section in order to report on each animals health and well-being.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Create a daily upkeep form for animals under their role  **Expectations**:   * Create and save a daily checklist form for animals in their section * Ability to create a form for each animal under their role/section. * Ability to edit the form in case of any mistakes | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all animals that are found in their role/section. | 1. Select Animal that they want to create the daily checklist form for. 2. Fill required form detailing all relevant infromation about the animal. 3. Save animal form and get redirected to the staff role UI. | 1. View all animal form that were created filtered by date. 2. Select the required record/form. 3. Select edit and change the infromation. |

## Staff User Experience Two

Staff Member filling out a daily food monitoring form in order to keep track to the animal’s food intake records that they have under their role.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Create a daily food monitoring form for animals under their role  **Expectations**:   * Create and save a daily form for animals in their section * Ability to create a form for each animal under their role/section. * Ability to edit the form in case of any mistakes | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all animals that are found in their role/section. | 1. Select Animal that they want to create the daily form for. 2. Fill required form detailing all relevant infromation about the animal’s eating habits. 3. Save animal form and get redirected to the staff role UI. | 1. View all animal form that were created filtered by date. 2. Select the required record/form. 3. Select edit and change the infromation. |

## Staff User Experience Three

Staff member filling a checklist fo the filters and pumps to ensure the equipment maintenanceis logged appropriately.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Create a weekly filter and pump checklist  **Expectations**:   * Create and save a weekly checklist for filters and pumps * Ability to fill out a checklist for maintanance * Ability to edit the checklist | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all the maintanance checklist under their role. | 1. Select the filter and pumps mainatanace checklist 2. View all the previous checklist filtered by recent 3. Create new checklist. 4. Fill out necessary details. 5. Save checklist and get redirected to the checklist view. | 1. View all maintanance checklist that were created filtered by date. 2. Select the required record/checklist. 3. Select edit and change the infromation. |

## Staff User Experience

Staff Member records the daily electric fence readings to ensure that the zoo can keep track of its integrety to security measures.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Create a daily reading of electric fences  **Expectations**:   * Create and save a daily reading for electric fences * Ability to fill out a reading for maintanance * Ability to edit the reading | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all the maintanance checklist under their role. | 1. Select the electric fence mainatanace tab. 2. View all the previous readings filtered by recent 3. Create new reading. 4. Fill out necessary details. 5. Save reading and get redirected to the maintanance view. | 1. View all maintanance readings under electric fences that were created filtered by date. 2. Select the required record/reading. 3. Select edit and change the infromation. |

## Staff User Experience Four

Staff member records the weather conditions for the day, so that they can note how external factors may impact the animals.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Logging the weather conditions daily  **Expectations**:   * Create and save a daily reading for the weather * Ability to fill out a reading for the weather * Ability to edit the reading | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all the weather readings under their role. | 1. Select the weather tab. 2. View all the previous readings filtered by recent 3. Create new reading. 4. Fill out necessary details. 5. Save reading and get redirected to the weather view. | 1. View all weather readings that were created filtered by date. 2. Select the required record/reading. 3. Select edit and change the infromation. |

## Staff User Experience Five

Staff member wants to submit a monthly fire extinguisher report, so that they can document the status of fire safety equipment.

|  |  |  |
| --- | --- | --- |
| Staff Member  **Scenario**: Creating a new report for fire extinguisher status  **Expectations**:   * Create and save a monthly status for the fire extinguishers * Ability to fill out a report * Ability to edit the report | | |
| Log In | Create Profile | Updating Records |
| 1. Log In to the Staff Role. 2. View all the fire extinguisher reports under their role. | 1. Select the maintanance tab. 2. View all the previous reports filtered by recent 3. Create new report. 4. Fill out necessary details. 5. Save report and get redirected to the maintanance view. | 1. View all the reports that were created filtered by date. 2. Select the required record/report. 3. Select edit and change the infromation. |

# Non-functional Requirements

**1. Performance**

* **Requirement**: The app must load the user's assigned section and related forms within 2 seconds of logging in.
  + **User Story**:
    - **As a staff member**, I want to log in and access my assigned section quickly, so that I can begin my daily tasks without delay.
  + **Acceptance Criteria**: After login, the app loads the staff member's dashboard and section within 2 seconds.
* **Requirement**: Form submission (e.g., food monitoring, checklist) must take less than 5 seconds after clicking submit.
  + **User Story**:
    - **As a staff member**, I want to submit forms with minimal delay, so that I can focus on my tasks efficiently.
  + **Acceptance Criteria**: Form submission completes in under 5 seconds.

**2**. Scalability

* **Requirement**: The app should support a maximum of 10 concurrent users without degradation in performance.
  + **User Story**:
    - **As an admin**, I want the app to handle all my staff members using it at the same time, so that operations aren't interrupted.
* **Acceptance Criteria**: The app should perform smoothly with 10 concurrent users submitting forms and accessing data simultaneously.
* **Requirement**: The database should scale to accommodate future additions of animal groups, camps, and forms without the need for architectural changes.
  + **User Story**:
    - **As an admin**, I want the app to easily handle more animal groups and data as the zoo grows, so that I don’t have to worry about performance issues.
* **Acceptance Criteria**: The app and its database can grow to support 50+ animal groups, species, and forms without requiring a change in architecture.

**3. Reliability Requirements**

* **Requirement**: The app must ensure 99.9% uptime to support daily zoo operations.
  + **User Story**:
    - **As a staff member**, I need the app to be available whenever I need to submit my daily reports, so that my work is not disrupted.
  + **Acceptance Criteria**: The app remains available 99.9% of the time during business hours, with minimal downtime for maintenance.
* **Requirement**: All data entered must be automatically saved, even in offline mode, and synced when the user regains connectivity.
  + **User Story**:
    - **As a staff member**, I want my form data to be saved even when I lose internet connectivity, so that I don’t lose work.
  + **Acceptance Criteria**: Data is stored locally if offline and synced automatically when connectivity is restored

**4. Maintainability Requirements**

* **Requirement: The app's codebase must be modular, allowing future developers to easily modify sections without affecting the entire system.**
  + **User Story:**
    - **As a developer, I want to make changes to one part of the app (e.g., adding a new animal group) without affecting other parts, so that maintenance is easier.**
  + **Acceptance Criteria: The code is modular, and changes to one feature (e.g., CRUD for animal groups) don’t break others (e.g., forms).**
* **Requirement: The app must support regular updates without causing downtime or data loss.**
  + **User Story:**
    - **As an admin, I want the app to be updated without interrupting daily zoo activities, so that work continues smoothly.**
  + **Acceptance Criteria: Updates can be pushed without causing service disruption or data loss.**

**5. Security Requirements**

* **Requirement**: Role-based access control (RBAC) must be implemented to ensure that staff only access their assigned sections.
  + **User Story**:
    - **As an admin**, I want to restrict staff members to their assigned sections, so that unauthorized users cannot access sensitive data.
  + **Acceptance Criteria**: Staff members are only able to access the sections assigned to them, based on their role.
* **Requirement**: All data must be encrypted in transit and at rest to ensure security and confidentiality.
  + **User Story**:
    - **As an admin**, I want to ensure that the data entered by staff is secure and protected from external threats.
  + **Acceptance Criteria**: All data is encrypted using industry-standard encryption methods.

**6. Usability Requirements**

* **Requirement**: The app’s interface should be simple and intuitive, with staff members able to complete tasks without requiring training.
  + **User Story**:
    - **As a staff member**, I want the app to be easy to use, so that I can quickly complete my tasks with minimal instructions.
  + **Acceptance Criteria**: New staff members should be able to navigate the app and submit forms without additional training.
* **Requirement**: Forms should be optimized for mobile devices, ensuring smooth navigation and input on small screens.
  + **User Story**:
    - **As a staff member**, I want to easily enter data into forms on my phone, so that I can complete my reports quickly.
  + **Acceptance Criteria**: Forms are designed for optimal mobile use with clear, easy-to-tap fields and buttons.

**7. Interoperability Requirements**

* **Requirement**: The app must integrate with third-party systems like weather APIs for daily weather recording and report generation.
  + **User Story**:
    - **As a staff member**, I want the weather data to be automatically pulled from external systems, so that I don’t have to manually input it.
  + **Acceptance Criteria**: The app retrieves weather data from an external API and integrates it into the daily reports.
* **Requirement**: The app should allow exporting of reports in formats like CSV or PDF for external use.
  + **User Story**:
    - **As an admin**, I want to export form data to external systems, so that I can store and analyse the information elsewhere.
  + **Acceptance Criteria**: Reports can be exported in CSV or PDF format.

**8. Internationalization / Localization Requirements**

* **Requirement**: The app must support multiple languages, including at least English and a local language (e.g., Afrikaans, Zulu), to accommodate the diverse staff.
  + **User Story**:
    - **As a staff member**, I want to be able to switch between English and Afrikaans, so that I can use the app in the language I’m most comfortable with.
  + **Acceptance Criteria**: The app includes language options for English and Afrikaans, with easy switching in the settings.

# Analysis artifacts

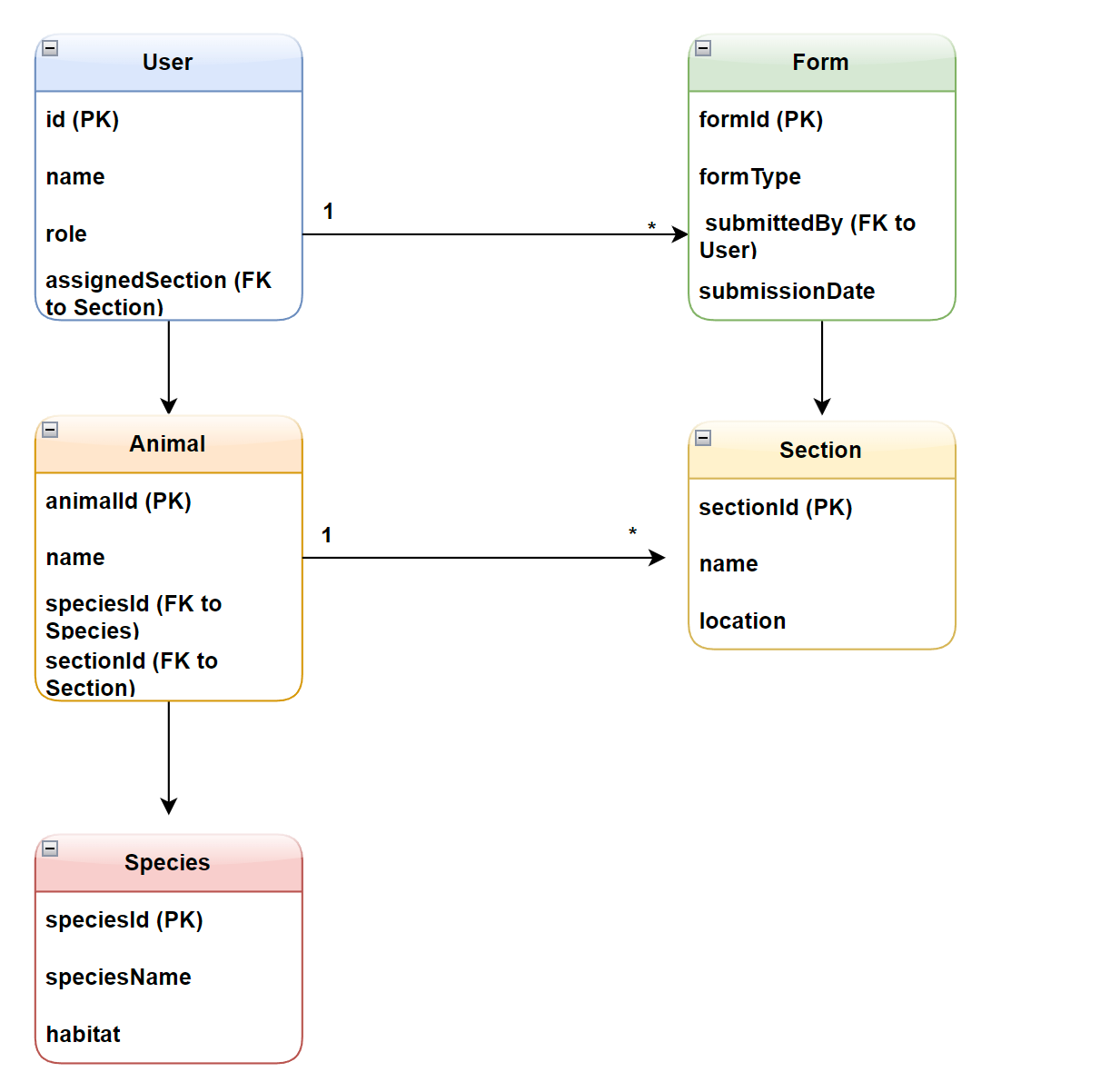
A diagram of a company

Description automatically generated with medium confidence**1. Description of Data Storage**

The data storage of the Lory Park Zoo app should be designed in such a way that each bounded context needs to store its data in a structured and organized way: User Management, Animal Management, and Form Management. Data will be persisted in a relational database; entities will be stored in tables corresponding to the objects described in the domain model.  
Every entity will be directly connected to a table in the database. Relationships between entities will be displayed through foreign keys and rules that ensure referential integrity.

**Example:**

* **User**: This entity will be stored in a table where each record corresponds to a user (Admin or Staff). This table will store attributes like the user's name, role, and assigned section.
* **Animal**: This entity will be stored in an animal table, capturing details like the animal’s name, species, and section.
* **Form**: This entity will be stored in a form table, representing various forms such as daily food monitoring or animal checklists. It will store data about who submitted the form and for which animal.



Relationship:  
  
 **User → Form**: A user submits multiple forms (one-to-many relationship).

**Animal → Section**: An animal belongs to a section (many-to-one relationship).

**Form → Animal**: A form is submitted for an animal (many-to-one relationship).

Explanation of Data Storage Using the ERD

In the domain model shown, the Lory Park Zoo app stores its data in ways that make it easy to create things that represent real-world objects. An ERD shows clearly how these things are arranged in the database and how they connect to each other.

1. Data Entities Match Domain Models

Each entity in the ERD corresponds directly to the objects described in the domain model, ensuring consistency between the application's design and its data storage. For example:

* User (Admin/Staff) matches the roles in the user management system.
* Animal represents the species and sections as described in animal management.
* Form corresponds to the various operational forms submitted by staff.

2. Logical Entity Relationship Diagram (ERD)

The ERD depicts the key data entities (e.g., User, Animal, Form, Section) and defines the relationships between them. It specifies:

* Primary keys to uniquely identify each entity.
* Foreign keys to establish relationships between different entities.
* Attributes that capture essential information for each entity (e.g., name, role, species, form submission date).

# Implementation DocumentationA screenshot of a computer screen Description automatically generated

**Sequence Diagram**

**Actors Involved:**

1. Admin: Manages staff, sections, and animal groups.
2. System: The core application that processes requests from Admin and Staff, interacting with the database.
3. Staff: Logs into the system, completes forms, and submits reports.
4. Database: Stores and retrieves information related to staff, sections, animal groups, and forms.

**Key Interactions:**

1. **Admin Creates Staff (CreateStaff())**:
   * **Admin** initiates a request to create a new staff member (CreateStaff()).
   * The **System** processes this request by updating the staff data in the **Database** (UpdateStaff\_DB()).
   * Once the staff data is successfully stored, the **Database** confirms the update (ConfirmUpdate()).
   * The **System** then sends a confirmation message back to the **Admin** (Confirm()).
2. **Assign Section to Staff (AssignSection())**:

* The **Admin** assigns sections (animal groups) to the staff (AssignSection()).
* The **System** updates the section assignments in the **Database** (UpdateSection()).
* The system confirms the update and sends a confirmation back to the **Admin**.

1. **Admin Creates Animal Group (CreateAnml\_group())**:

* The **Admin** creates or updates animal groups (CreateAnml\_group()).
* The **System** processes the update and stores it in the **Database** (UpdateAnimalGroups()).
* The **Database** confirms the change (ConfirmUpdate()), and the **System** notifies the **Admin** that the action was successful (Confirm()).

1. **Staff Login (Login())**:

* A **Staff** member logs into the system (Login()).
* The **System** processes the login request and retrieves the assigned section for the staff member (StaffSection()), ensuring the staff is directed to the correct area of the application.

1. **Staff Submits Form (SubmitForm())**:

* The **Staff** member submits a form, such as a daily checklist or a monitoring sheet (SubmitForm()).
* The **System** stores the form details in the **Database** (UpdateForm\_DB()).
* The **Database** confirms that the form has been saved (ConfirmSave()), and the **System** sends a confirmation back to the **Staff**.

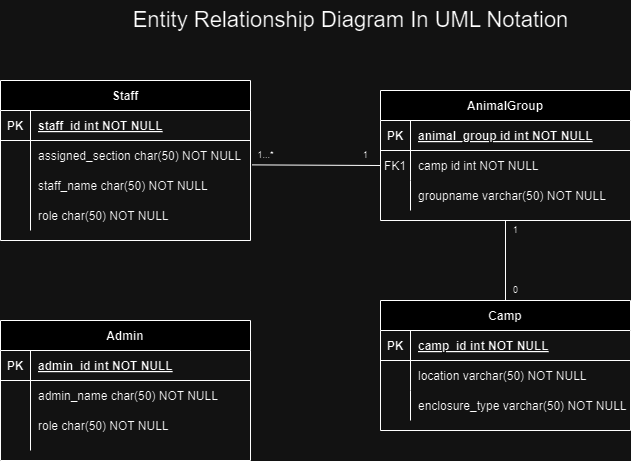
1. **Admin Manages Staff Reports (ManageStaffReports())**:

* The **admin** initiates the process of managing or reviewing staff reports (ManageStaffReports()).
* The **System** retrieves the relevant report from the **Database** (RetrieveRecord()).
* Once the report is retrieved, the **Database** sends the report data to the **System** (RetrievedReport()), and the **admin** is notified.

1. **Admin Updates Staff Reports (UpdateSaveChanges())**:

* If the **Admin** makes any changes to the reports, they submit the updated report (UpdateSaveChanges()).
* The **System** processes the changes and updates the data in the **Database**.
* After updating the database, the **System** confirms that the changes have been successfully saved (Confirm()).

# Data Scheme Documentation



**Entity Descriptions:**

1. **Staff**: Stores staff details, including their assigned section (e.g., reptiles, primates). Each staff member has a unique ID (StaffID), a role (admin or general staff).
2. **Animal Groups**: Represents a group of animals under the care of staff. Each group has a unique AnimalGroupID, a GroupName (like "Big Cats" or "Reptiles"), and a reference to a **Camp** (enclosure) where they are located.
3. **Camp**: Represents the physical enclosure where animal groups are housed. It stores CampID, the **location** of the enclosure, and the type of enclosure (EnclosureType).
4. **Admin**: Represents the admin entity with details like AdminID, their **name**, and their **role** (which could include managing staff, animal groups, and viewing all reports).
5. **Forms (Not in ERD)**: Though not modeled in the ERD, **Forms** like daily checklists, monitoring sheets, and food logs are stored in relation to **Staff** and **Animal Groups**.

The **relationships**:

* A **Staff** member can manage one or more **Animal Groups**.
* An **Animal Group** is assigned to a specific **Camp** (enclosure).
* **Admins** can access all entities and forms submitted by the staff.

**Firestore Data Structure**

Since Firestore is a document-based NoSQL database, data is organized into collections and documents. Below are the main collections and subcollections in your app, along with their corresponding JSON schema.

**1. Staff Collection**

The Staff collection stores the user data for staff members (including admin), along with their assigned sections, roles, and login information.

JSON Schema (Staff Document):

{

"staffID": "unique\_staff\_id\_123",

"name": "Staff Name",

"email": "staffname@example.com",

"assignedSection": "Big Cats",

"role": "staff",

"createdAt": "Date&Time "

}

**2. AnimalGroups Collection**

Each animal group is stored in a separate document within the AnimalGroups collection. The group document links to its specific camp (enclosure).

**JSON Schema (Animal Group Document)**:

{

"animalGroupID": "group\_001",

"groupName": "Big Cats",

"species": [

{

"speciesName": "Lion",

"animalName": "Simba",

"healthStatus": "Healthy"

},

{

"speciesName": "Tiger",

"animalName": "Shere Khan",

"healthStatus": "Injured"

}

],

"campID": "camp\_002"

}

**3. Camps Collection**

Camps (enclosures) store information about each location, like the type of enclosure and its physical location in the zoo.

**JSON Schema (Camp Document):**

{

"campID": "camp\_002",

"location": "North Section",

"enclosureType": "Open Air"

}

**4. Forms Collection**

Each form submitted by staff (e.g., daily food monitoring, animal health checklists) is stored in a subcollection under the Staff or AnimalGroups collection. This ensures each form is tied to both the staff members and the relevant animal group.

**JSON Schema (Food Monitoring Form Document):**

{

"formID": "form\_001",

"submittedBy": "staffID\_123",

"section": "Big Cats",

"feedingData": [

{

"animalName": "Simba",

"timeFed": "08:00 AM",

"portion": "1/4"

},

{

"animalName": "Shere Khan",

"timeFed": "08:15 AM",

"portion": "3/4"

}

],

"submittedAt": "2024-01-12T10:00:00Z"

}

# Architecture artifacts

## Design Patterns

**1.1 Singleton Pattern**

* The application has used the Singleton Pattern in certain parts that always require, during its lifetime, to have just one single instance of the class, like DatabaseHelper, which uses the SQLite database. In doing so, it makes sure not to utilize too many resources by creating multiple connections to the database, minimizing potential conflicts or overloads in memory.
* Justification: The Singleton Pattern restricts the database connections and allows, or enforces, a single point of access for all database operations, quite suitable for applications that demand centralization of data access and management.

**1.2 Repository Pattern**

* The Repository Pattern is used to handle the access to data by abstracting the details of data storage and the API of data management in a clean way, especially for entities like User, Animal, and Form.
* Justification: This pattern decouples the database layer from the business logic, and therefore makes the code easier to maintain and test. Besides that, it promotes flexibility since changes in the data source do not hinder other parts of the application.

**1.3 Factory Pattern**

* The Factory Pattern is used in instantiating objects, such as forms like food monitoring and animal checklists, according to specific needs. The application can dynamically create different types of forms by using the factory.
* Justification: The Factory Pattern makes the instantiation of objects easier and encapsulates the cumbersome logic. This is useful when creating instances of different types of forms because this allows adding or modifying forms without changes in the code

## Architecture Patterns

**2.1 Model-View-Controller Pattern**

* In the Lory Zoo application, MVC forms the basis of the division of labour between the UI, business logic, and data access layers:
* Model - Represents data and process the business rules. Examples: entities like User, Animal, and Form.
* View - Comprises UI components with which the staff interact, for example, forms on which data entry is made, or reporting.
* Controller - Manages inter-resource communications between model and view, handling any user input and updating the View as appropriate to react to that input.
* Justification: MVC offers clear separation of concerns, hence improves maintainability and readability. It will also facilitate parallel development by letting the user interface, business logic, and data access layers evolve independently from each other.

**2.2 Microservices Architecture (Cloud Components)**

* In the cloud environment, the application is broken down as a microservices architecture, which is a collection of independent loosely coupled services. Each service will take only one responsibility of User Management Service, Data Storage Service, and Form Management Service.
* Justification: Microservices are scalable and reliable. Since every one of the services can be individually scaled, this enhances resource utilization. This will also allow independent updates or changes in one service without influencing other services

## Cloud Architecture

**3.1 Cloud Infrastructure**

* The Lory Park Zoo app relies on cloud infrastructure provided by Firebase to handle its storage, authentication, and hosting needs. Firebase provides scalable services to meet the app's requirements for data storage, real-time updates, and secure user authentication.
* Components:
* Firebase Authentication: Provides user authentication, covering safe login and registration, and role-based access control.
* Firebase Database: Houses user information, records of animals, forms, and logs. It also supports real-time data synchronization.
* Firebase Hosting: it hosts the application backend on a secure and scalable platform for API if and Web Dashboard needed.

**3.2 Cloud Architecture Decisions**

* The reason the selection has fallen on Firebase is that it is highly scalable, easy to integrate, and natively supports mobile applications with no extensive backend development.
* Firebase will act in place of traditional SQL databases for better flexibility on the handling of unstructured data and real-time updates, best suited for form and report data.

**3.3 Cloud Networking and Protocols**

* Networking: Firebase's implementation of networking allows secure data in transit from the client to the server by automatically managing network loads and load balancing.
* Protocols: HTTPS will be used for service communications using SSL/TLS encryption. This makes any data in transit not easily intercepted.
* Justification: HTTPS and SSL/TLS encryption ensure that data transferred between clients and Firebase services is secure, hence protecting user privacy and app data integrity.

**3.4 Cloud Security**

* Role-Based Access Control (RBAC): This enforces role-based permissions through Firebase Authentication. The users will be restricted to sections of the application depending upon their role such as Admin and Staff.
* Data Encryption: Data encryption will be ensured both during transportation with SSL/TLS and at rest in Firebase, according to the industrial standard for data security.
* Auth tokens are generated by Firebase to authenticate user sessions and provide additional security by the validation of tokens that only those that are authenticated can have access to certain services.

## Data Structures and Algorithms

**4.1 Data Structures**

* Dictionaries - to store key-value pairs of form data for quick lookups and proper management of form attributes
* Queues - in the background tasks, like in synchronization with the server; ensures that tasks are processed in order, and it supports consistency in data.
* Sets - Maintenance of unique lists, such as unique animal species or sections to avoid data duplication

**4.2 Algorithms**

* Sorting Algorithms: These have been used in the presentation of forms and reports in chronological or alphabetical order.
* Search Algorithms: Implemented to perform quick locating of animal records or user information for efficiency in the retrieval of data. These structures and algorithms have been chosen for the optimization of performance of the app's data management and for assuring smooth processing of data in real time, according to industrial best practices.

## Data Transfer and Integration

* REST APIs: RESTful APIs are used for communicating well between the frontend and the backend, to make sure data is correctly and efficiently passed between services.
* Third-Party Integration
* Weather API: utilized in capturing daily weather conditions programmatically and enhances efficiency during reporting.
* Firebase Authentication API: This was employed in the verification of a user and the handling of secure logins and sessions.
* Justification: REST APIs provide a standardized communication layer that enables flexibility and modularity in the application, thus allowing future integrations with other services or APIs.

# Security

Security is a critical aspect of the Lory Park Zoo app, as it handles sensitive operational data about animals and staff. Here’s how security is addressed, following the criteria provided:

1. Potential Threat Actors

* Threat Actors: Any individual or organization that may potentially exploit system weaknesses. The following are potential threat actors for the Lory Park Zoo app
* External Attackers: These are individuals or groups attempting to break into the application unauthorized, especially to access sensitive staff and animal data.
* Insider Threats: Disgruntled employees or individuals with permission using their access for wrongdoing.
* Unintentional Threats: Users who might accidentally put security at risk by doing things poorly (like using weak passwords or unsecured devices)

2. Potential Threat Vectors

* Threat vectors refer to the means in which an attacker may leverage the weaknesses.

Unauthorized Access:

* Users might attempt to access areas or information with which they should not concern themselves, especially if proper RBAC is not implemented.

Data Interception:

* Transmission between mobile devices and servers will leak information when the transfer contains sensitive information, like staff duties and details about the animal, if it is not encrypted.

Data Manipulation:

* The staff can change various forms and reports without proper security.

Phishing and Social Engineering:

* The users are tricked to provide their login information or click on various malicious links.

3. Mitigations for Threats

* Above threats are addressed by putting in place security measures that ensure the economy of mechanism, where controls applied offer proper and effective detection.
* Access Control based on Role:  
   Only allowed users (like admin or assigned staff) can get into certain parts of the app.

Only staff can process forms and reports for their assigned animal groups.

* Economy of Mechanism: RBAC simplifies permission enforcement using predefined roles.

Data Encryption:

Data in Transit: Because SSL/TLS is used, all data transit between the devices and the server will be encrypted against possible intercepts.

* At Rest: Sensitive data within the database will be encrypted, such as user credentials and animal records.
* Economy of Mechanism: Use of SSL/TLS reduces the complexity of making communication secure.

4. Balancing Security with Usability

A major challenge to security involves finding a balance between tight security rules and usability. This will help staff perform their jobs well without too many problems.  
  
RBAC: By granting certain roles and permissions, this application ensures that the staff see only what is relevant. This balances out security with usability, as staff will not be overwhelmed with nonsensical information or forms.

* User Interface Simplicity: The app is made to keep the UI simple and easy to use, so staff can quickly submit forms while security checks happen in the background.
* MFA: Optional. Multi-Factor Authentication can be used by admin users to make their accounts more secure without inconveniencing regular staff during login.
* Offline Mode: Also, there were security plans for offline access, ensuring data is secure even as staff goes offline; when going back online, data safely syncs.

5. Complete Mediation

Complete mediation means to check every request to sensitive data against the security rules - e.g., RBAC. In the application:

* -User actions, such as form submit or animal data view, are controlled by their role. This blocks unauthorized access.
* Everything that interacts with the app is logged for an audit trail in suspicious activities.
* Encryption ensures that the sensitive data that users access remains secure during and after handling.

Conclusion

The Lory Park Zoo app enforces a full security suite that addresses potential threat actors, vectors, and mitigations. Using RBAC, encryption, input validation, and secure authentication, the app reaches a balance of security versus usability, making it a secure yet efficient operational tool for zoo staff.

# DevOps

To address the non-functional requirement of rapid deployment in our project, our team has implemented a CI (Continuous Integration) pipeline using GitHub Actions. This pipeline automates several key steps in the development process, ensuring high quality and efficiency.  
Below is a detailed description of the functionality of your GitHub Actions pipeline, along with a diagram to illustrate its workflow.

## GitHub Actions Pipeline overview

The pipeline is triggered on two main events: when there is a push to the main branch and when a pull request is made to the main branch. It consists of several jobs, each responsible for specific tasks, which are executed sequentially based on dependencies.

**Jobs in the Pipeline**

1. **Install Dependencies**
   * **Purpose:** Sets up the environment and installs project dependencies.
   * **Steps:**
     + Checkout the code from the repository.
     + Set up Node.js version 18.
     + Run npm install to install dependencies.
2. **Run Linting**
   * **Purpose:** Ensures code quality by checking for linting errors.
   * **Dependencies:** Needs the install job to complete first.
   * **Steps:**
     + Checkout the code.
     + Set up Node.js.
     + Install dependencies again (necessary for linting).
     + Run ESLint with npm run lint.
3. **Run Prettier**
   * **Purpose:** Formats code according to predefined styles to maintain consistency.
   * **Dependencies:** Needs the install job to complete first.
   * **Steps:**
     + Checkout the code.
     + Set up Node.js.
     + Install dependencies.
     + Run Prettier with npm run prettier --check ..
4. **Build Android App**
   * **Purpose:** Compiles the Android application.
   * **Dependencies:** Needs the install job to complete first.
   * **Steps:**
     + Checkout the code.
     + Set up Node.js.
     + Install dependencies.
     + Set up Expo Application Services (EAS) using a secret token.
     + Build the app with eas build --profile development --platform all --non-interactive.
5. **Run Jest Tests**
   * **Purpose:** Executes unit tests to validate code functionality.
   * **Dependencies:** Needs the install job to complete first.
   * **Steps:**
     + Checkout the code.
     + Set up Node.js.
     + Install dependencies.
     + Run tests using npm test.

## Pipeline Workflow Diagram

Github Actions Workflow


# Running costs

**1. Firebase Costs (ZAR)**

* **Authentication**: Free (up to 10 users)
* **Cloud Firestore**: Basic usage estimated at ~50,000 reads, ~10,000 writes, and minimal storage
  + Reads: ~R20
  + Writes: ~R5
  + Storage (~1 GB for forms and logs): ~R40
* **Hosting**: Assuming the app uses Firebase Hosting for the web dashboard or API interactions
  + Estimated: R35 (Free tier covers small traffic, additional cost applies beyond that)
* **Cloud Functions** (if background processing is used):
  + Estimated cost: ~R50 for lightweight functions

**2. Apple Developer Program Subscription**

* Annual subscription: **R1,700/year**
  + Monthly equivalent: R142pm

**3. Miscellaneous (Optional Services)**

* **Weather API** integration (assuming limited API calls):
  + Free tier or ~R50/month (depending on service).
* **Expo EAS Services (for building iOS/Android apps)**:
  + Free tier sufficient for 10 users.

|  |  |
| --- | --- |
| Service | Monthly Cost (ZAR) |
| Firebase (Firestore, Hosting, Functions) | R150z |
| Apple Developer Program (Monthly Equivalent) | R142 |
| Weather API (Optional) | R50 |
| Total Estimated Cost | **R342** |

**Growth Projections (2 Years)**

* **Year 1**:
  + Costs remain stable as the app is limited to 10 users.
  + Occasional scaling in Firebase usage may occur (up to R50 more per month due to increased data).
* **Year 2**:
  + Assumption: Small increase in data storage (~3-5GB).
  + Firebase total may increase by R50-100 per month.

**Estimated monthly cost after 2 years**:

* **R400-R450/month** considering that the forms and the data that the zoo logs, do not change.

# Change Management

# Appendices

# References

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley. [Online]. DOI: 10.5555/2005498 [Accessed 21 October 2024].

Martin, R.C. 2017. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Prentice Hall. [Online]. Available at: https://www.pearson.com/store/p/clean-architecture/P100000711093 [Accessed 20 October 2024].

Evans, E. 2004. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Addison-Wesley Professional. [Online]. Available at: https://www.domainlanguage.com/book/ [Accessed 21 October 2024].

Fowler, M. 2002. **Patterns of Enterprise Application Architecture**. Addison-Wesley. [Online]. Available at: https://martinfowler.com/books/eaa.html [Accessed 21 October 2024].

Firebase. 2024. **Firebase Documentation: Authentication, Firestore, and Hosting**. Google Firebase. [Online]. Available at: https://firebase.google.com/docs [Accessed 23 October 2024].

Goyal, D. 2021. **Microservices Patterns Explained**. DigitalOcean. [Online]. Available at: https://www.digitalocean.com/community/tutorials/microservices-patterns [Accessed 25 October 2024].