COS301

WILDLIFE DRONE FLIGHT PLANS SYSTEMS REQUIREMENTS SPECIFICATION

DR BAM

Matthew Evans	16262949
Andreas Louw	15048366
Bryan Janse van Vuuren	16217498
Deane Roos	17057966
Reinhardt Eiselen	14043302

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to present an overview the proposed drone flight plan system. To elicitate its requirements, along with more information on the system parameters and goals. To describe the target audience, user interface, and high-level hardware and software requirements for the proposed system. This document is intended for EPI-USE, the stakeholders, and the developers who will implement the system.

1.2 Vision

Our vision is to create a real world applicable system that in the future will aid in the conservation of rhinos and elephants.

1.3 Objectives

- To create a prototype of this system that creates randomized optimal fight plans for drones.
- To increase the effectiveness of drone patrols by providing them with a more extensive overview of flight plans and map activities.
- Empower rangers with information about poaching areas within reserves and creating routes to patrol based on this information.

1.4 Business needs

To allow for more optimal and accurate route creation so that the drones can cover more key areas to help prevent the poaching of rhinos and elephants. While randomizing routes to prevent the possibility of patterns being recognized by poachers.

1.5 Scope

The proposed system is a flight plan system that will generate random optimal flight plans for drone pilots and patrol routes for rangers based on poaching hotspots. Pilots and rangers will be notified of the routes and changes to them.
Hotspots will be generated taking into account past incidents, geographical information and animal tracking data.
Flight plans will be generated by taking in the drones capabilities and the hotspots at the time as well as historical data and current incidents reported by rangers and pilots. Rangers and pilots will be able to indicate points of interest on the map.

2 Domain Model

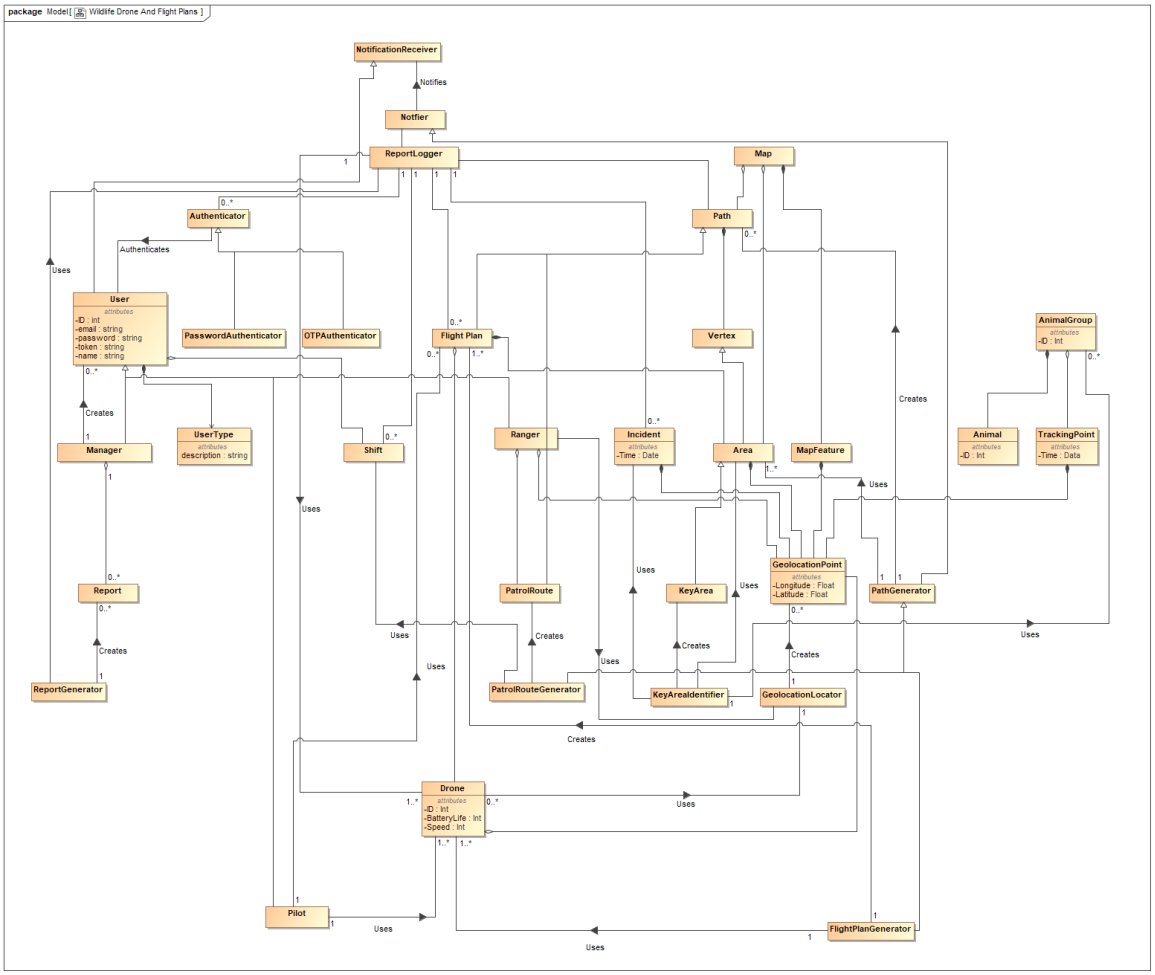
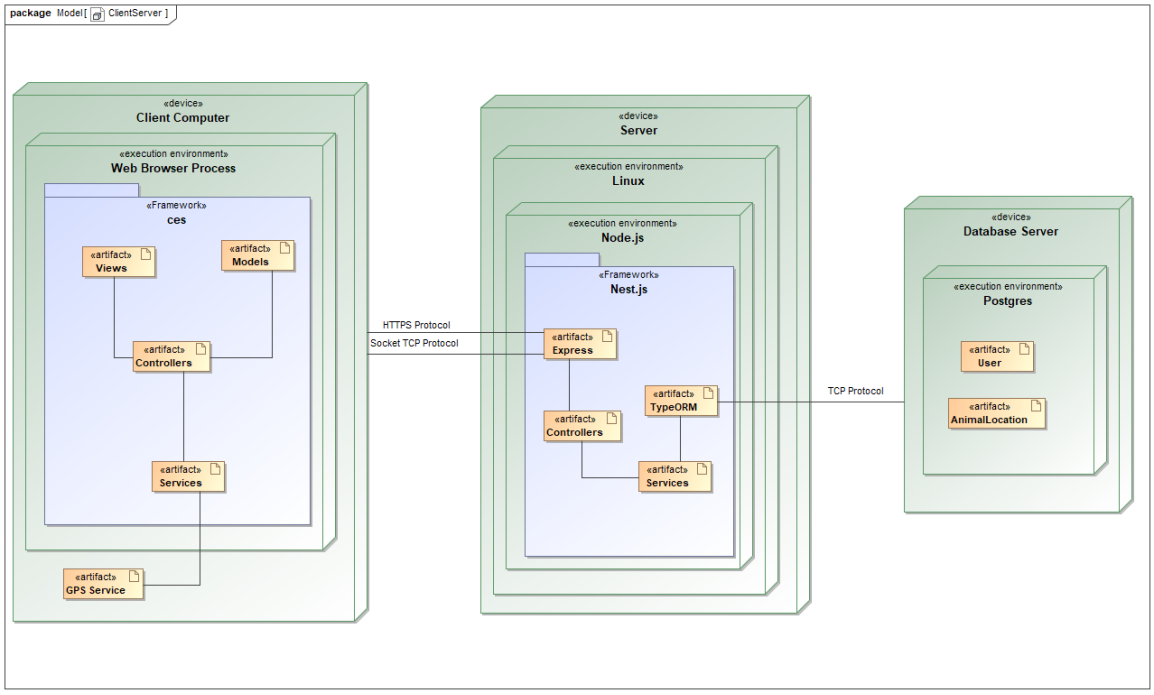


Figure 1: Back-End System.

3 Deployment Model



4 Architectural design

4.1 System as a whole

The overall system follows a client-server architecture. The system requires a high level of security and secrecy due to the nature of poaching, and, as such, all processing will be done server side (otherwise the client could be reverse engineered). Anti-poaching data is considered very sensitive, and this method allows all data to be stored server side, and only results of computation to be provided client side.

Most communication between the client and server uses the HTTPS protocol. This provides a simple and secure protocol catered to requests and responses – perfect for the client to call the API. Sockets will be used to communicate live data between the client and server – such as current device geolocation.

The client-server architecture also provides a relationship where the clients need not know each other – this enables the client to be written in a multitude of languages/frameworks for different operating systems, and they will all work so long as they adhere to the interface of the server. This will make cross-platform development easier.

Another advantage to client-server architecture is processing can be offloaded from the client devices to the server, which will typically be mobile devices where battery life is critical to the user. This will result in a better user experience. A drawback to this approach is that the server and client need to have constant communication in order to work – as such, a caching layer is implemented to provide limited offline functionality to the client in the case of network or server error (discussed later).

On the server side, an N-tier architecture is used with Model View Controller. The layers are implemented as a connection/thread pool, middleware, routes, and a persistence layer. This provides a modular architecture where parts of the system are interchangeable and easily modified thanks to low coupling between the layers. The connection pool is handled by Node.js and Express.js, providing an efficient server that can handle multiple concurrent clients. The middleware is handled by Nest.js, and provides an abstraction over the API endpoint routes to perform reusable actions such as authentication and error handling.

The server follows MVC, because the model is realised as a persistence layer, the controller as routes and services, and the view as the API's JSON REST interface. Services are associated with controllers via dependency injection, making them reusable and easily replaceable (low coupling). Services can therefore be designed with high cohesion in mind, adhering to the single responsibility principle.

Client-side, Model View Controller is also used in an N-tier architecture. The view is an HTML and CSS representation rendered using Cordova or the web browser. The controller is realised using controllers and services. As with the server side, dependency injection is used and provides the same benefits as for the server. The model is implemented as a cache between server calls and a persistence layer.

4.2 Subsystems

Map System
The map subsystem will be an interactive subsystem where the user will have to login to the system, this will allow the user access to the map. The n-tier architecture style will be used.

Hotspot identification subsystem
The hotspot identification subsystem will be a heuristic problem solving system where the hotspots that will be identified will use various data from the system these include but not limited to: past poaching incidents, probability of animals in area, probability of poaching, interest points allocated by users. This data will be used to predict possible poaching hotspots that may occur in future. The blackboard architectural style will be used.

Animal prediction subsystem
The animal prediction system will be a heuristic problem-solving system where the location of an animal will be predicted for an x amount of time in the future. The future location of the animal will be predicted using varies data that include but is not limited to: past location, habitat, temperature, weather, distance to water. The blackboard architectural style will be used.

Route generation System Route generation subsystem will be an interactive subsystem where the user will initiate a new route to be generated, the system will respond with the new route shown on the map. The N-tier architectural style will be used.

Data modification subsystem
The data modification subsystem will be an interactive and an object-persistence system; it is an interactive system by allowing the user(administrator) or other subsystems to add, remove and modifying data that is stored in the database this will have to be initialised by the user to start the process.. The system is also an object-persistence system by allowing the for storing and retrieving objects from the database and file systems. The system will mainly use n tier architecture with a persistence framework architecture on a lower tier of the n tier architecture.

Geolocation subsystem
The geolocation subsystem will be an event-driven system as a device (smartphone or drone) will ping the system and send the current location to the system where the system will then relay the data to the server. An event-driven system architecture will be used.

Notification subsystem
The notification subsystem will be an event-driven system that will be triggered by other systems such as the data modification, route generation and other systems by sending the notification system data and the notification system will then process the data accordingly. An event-driven system architecture will be used.

Authentication subsystem
The authentication system will be an interactive system as the user has to interact with the system to log in the data will then be processed by the authentication system to either allow access or deny access. A n-tier architecture will be used.

Reporting subsystem
The reporting system will be an event-driven system as other system will send data to the reporting system, the system will then process the data and send the data to the data modification system. An event-driven system architecture will be used.

5 User Characteristics

5.1 Drone Pilot

Will be piloting the physical drone. Will interact with the system to determine flight plan and routes that need to be traveled within a shift. These routes will be restricted to the game reserve's airspace. A drone pilot will also be able to use to use the system to add new hot spots or to report incidents.

5.2 Patrol Rangers

Will interact with the system to determine the patrol route to follow within a shift. These routes will be restricted to reserve roads. A ranger will also be able to use the system to report incidents and to add hot spots.

5.3 Administrators

Will be maintaining and updating the system as necessary. Will also have separate view from ranger and pilots to run diagnostics on data gathered.

6 Technology decisions

We have primarily focused on using free, open-source technologies which are highly configurable and provide cross-platform support.

The system as a whole uses Typescript. Typescript transpiles to Javascript, providing better features such as classes, inheritence and static typing. Javascript is used both server-side and client-side.

Node.js is used as the server's execution environment. This executes the project code efficiently, and provides system-level APIs such as filesystem interaction. Node provides cross-platform support for many system architectures and operating systems, making it easy to change hardware. Node is also widely supported as a cloud-based platform.

Nest.js provides a framework that abstracts Express.js on top of Node.js into a more modular system similar to Angular. It enforces splitting of code into sections similar to Model View Controller, and as such provides low coupling and high cohesion. This also automatically provides thread pooling for incoming connections.

Nest.js also provides dependency injection, making it easier to Passport.js is used to provide industry-standard authentication protocols. Because of how ubiquitous it is, it is easy to find modules for additional functionality such as two-factor authentication, which can then be quickly applied to its modular structure.

TypeORM is used to provide a persistence layer for the server. TypeORM is designed specifically to work with TypeScript, and uses classes with decorators to achieve its object relational mapping - a technique native to TypeScript. This provides an interface that abstracts the actual database usage into purely Typescript (no queries), which allows easy plugging in of multiple database providers. TypeORM supports a large variety of database providers.

Though the system has been designed such that database providers can be interchangeable, the recommended provider is Prostgres. Postgres is free and open source, and widely provided as a database option on cloud platforms. It also has been deisgned to optimise geographic data, which will provide enhanced performance for our system.

Testing is performed using Jest. This works well and provides built in testing functionality to make tesitng easier. It also makes it relatively easy to exchange for another unit testing framework in future.

The client uses Ionic framework. This allows the use of web technologies, such as Typescript, (which the majority of the development team is familiar with) to create mobile applications that closely mimic native design systems (it can also create desktop web applications). This reduces

the time spent on implementing visual features. Ionic is built on Cordova, which is supported by Apache. This allows a single mobile application to be developed as a progressive web app, then from the same codebase deploy for Android, iOS, the mobile web and desktop web. This reduces code repetition, and as such minimises bugs and development time.

Ionic is built on top of Angular. This provides a Model View Controller based framework that provides dependency injection. This provides separation of concerns and helps enforce the single responsibility principle. Dependency injection makes the system incredibly modular, and provides re-use of components.

Leaflet.js provides an interface for creating user-interactable maps in web applications. It is free and open source, and widely used and supported. It is relatively performant and works well with touch interfaces. It also works with many standard GIS formats.

OpenStreetMaps is used for gathering map data. It uses the Overpass API. This provides a queryable API to efficiently download filterted OpenStreetMaps data. This is a free service, and provides relatively accurate open source information when compared to paid alternatives. An advantage to this is that it can be edited by system users to provide better accuracy of the system.

Google Maps' satellite tiles are used for satellite imagery. These images are freely available with relatively high resolution (up to 30cm accurate). The images also provide contextual overlay information based on the location of map features (such as road names).

7 Functional Requirements

7.1 Use Cases

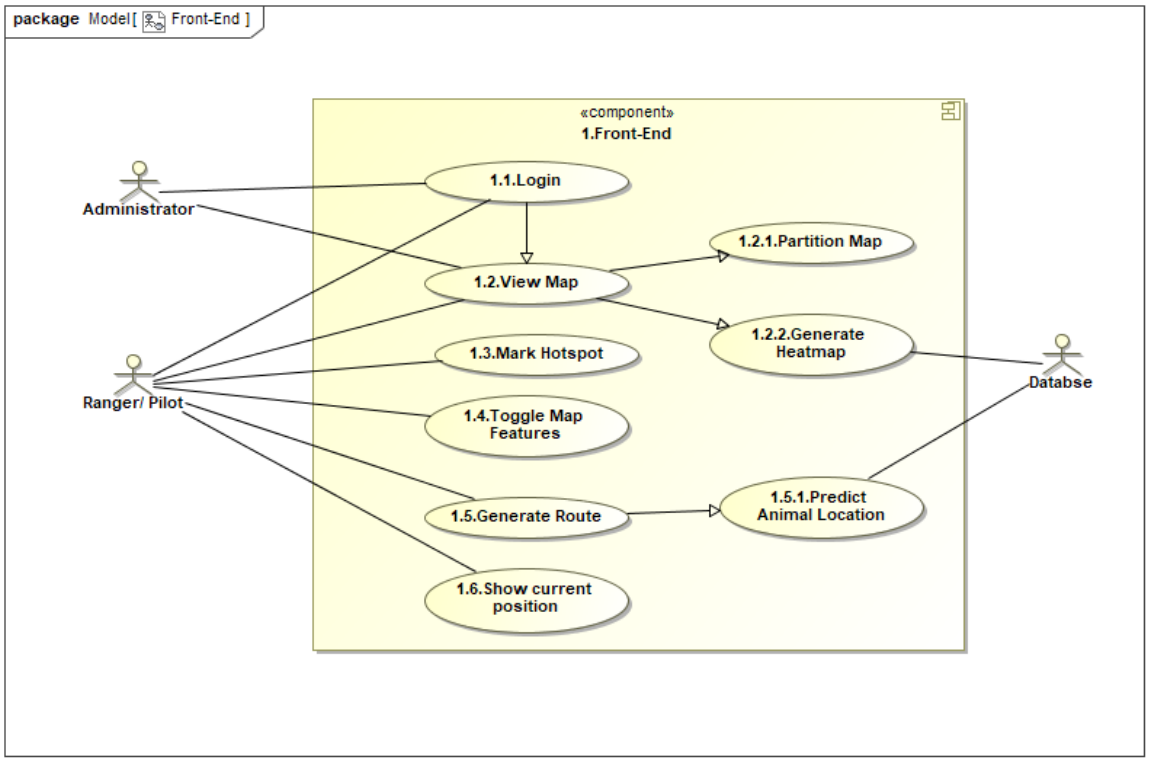


Figure 2: Front-End System.

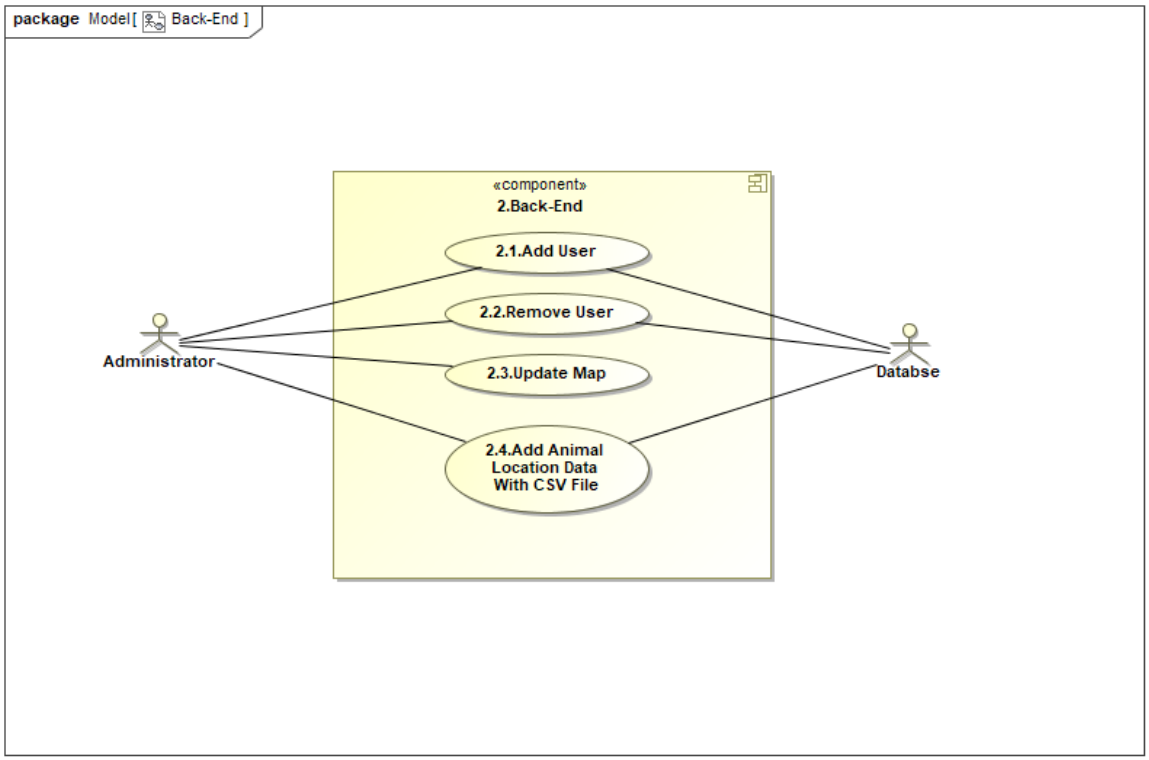


Figure 3: Back-End System.

7.2 Requirements

R1: The system will provide a map for the user.

- **R1.1:** The system will provide an overlay of routes on the map
  - **R1.1.1:** The system will provide an overlay of pilot routes on the map as generated by **R2.1**.
  - **R1.1.2:** The system will provide an overlay of ranger routes on the map as generated by **R2.2**.
- **R1.2:** The system will provide points of interest on the map
  - **R1.2.1:** The system will show user marked points on the map as used in **R4**.
  - **R1.2.2:** The system will allow ranger locations to be shown on the map as gathered from **R3.1**.
  - **R1.2.3:** The system will allow drone locations to be shown on the map as gathered from **R3.2.1**.
- **R1.3:** The system will provide animal hot-spots on the map **R7.1**.

R2: The system will generate random optimal routes.

- **R2.1:** The system will generate a flight plan for drones.
- **R2.2:** The system will generate a possible patrol route for rangers.

R3: The system will track user devices

- **R3.1:** The system will track ranger device location.
- **R3.2:** The system will track drones.
  - **R3.2.1:** The system will track the drone location.
  - **R3.2.2:** The system will track the drone systems, such as battery life.

R4: The system will allow for points of interest to be marked on the map

- **R4.1:** The system will allow pilots to mark points of interest.
- **R4.2:** The system will allow rangers to mark points of interest.

R5: The system will provide notifications for rangers and drone pilots.

R5.1: The system will notify drone pilots when a ranger added a point of interest.

R5.2: The system will notify rangers when a drone pilot added a point of interest.

R5.3: The system will notify pilots when a new flight plan has been generated

R5.4: The system will notify rangers when a new patrol route has been generated

R6: The system will store historic data

R6.1: The system will store previously used flight routes

R6.2: The system will store previously used patrol routes

R6.3: The system will store animal migration data

R7: The system will handle historic data

R7.1: The system will create heat-maps of animal locations based on historical data from **R6.4** and live data(If available).

R7.2: The system will show previous flight plans

R7.3: The system will show previous patrol routes

R7.4: The system will predict migration patterns of animals based on **R6.4**

R8: The system will provide user authentication to use the system

R8.1: The system will allow users to log in

R8.1.1: The system will authenticate log in details

R8.1.2: The system will inform a user of whether authentication was successful or not

R8.1.3: The system will generate a one-time pin

R8.1.4: The system will send users a one-time pin

R8.1.5: The system will authenticate a one-time pin

R8.2: The system will allow users to log out

R8.3: The system will allow administrators to manage other users

R8.3.1: The system will allow administrators to register other non-administrator users

R8.3.2: The system will allow administrators to disable other non-administrator users

R8.3.3: The system will allow administrators to change user passwords

R8.3.4: The system will hash passwords

R9: The system will provide reporting of activities.

R9.1: The system will provide logging of user log in attempts.

R9.2: The system will provide logging of points of interest done by rangers and pilots.

R9.3: The system will provide logging of routes taken by users.

R9.4: The system will provide logging of incidents.

R9.5: The system will generate reports from logs for administrators.

7.3 Subsystems

7.3.1 Map subsystem

The map subsystem is responsible for displaying map data to the user. It handles loading map data, visually representing it, and handling user interaction such as moving the map and zooming.The map will allow users to toggle between different layers and or features. An example would be a ranger able to view the flight plan of the drone and vice versa. Users will also be allowed to mark hotspots/points of interest onto the map

7.3.2 Hotspot identification subsystem

The hotspot identification subsystem is responsible for identifying hotspots from map data, past poaching incidents, and animal locations.These Hotspots will be used in the generation of routes for Rangers and Drone pilots to help generate more relevant routes to patrol.

7.3.3 Animal predictions subsystem

The animal predictions subsystem is responsible for predicting the future location of animals from past tracking information and map data.The result from processing this information will be used in the generation of routes.The aim of this subsystem it to increase effectiveness of route generation

7.3.4 Route Generation Subsystem

The route generation subsystem is responsible for finding an optimal "travelling salesman" circuit through a set of given hotspots.The system will provide routes for both the Ranger patrols routes and Drone pilot routes.

7.3.5 Data modification subsystem

The data modification subsystem handles the creation, updating and deletion of park data, allowing users to change map features, drone information, add animal tracking data and add hotspots.

7.3.6 Geolocation subsystem

The Geolocation subsystem is responsible for locating the user's device using device by periodically pinging the Geolocation services of the mobile devices. The subsystem relays this information back to the server for processing.

7.3.7 Notification subsystem

The notifications subsystem is responsible for sending notifications to all designated users.

7.3.8 Authentication subsystem

The authentication subsystem is responsible for handling authentication of users, including the creation and managing of users. It is also responsible for verifying two-step authentication.

7.3.9 Reporting subsystem

The reporting subsystem is responsible for handling logging of data generated by and used by users. The subsystem will generate reports for the administrators.

8 Constraints

- Users using the system are either a administrator, pilot or ranger of the reserve using the system.
- The system will have a stable internet connection (such that timeouts do not occur).
- The mobile application will be run on a device with sufficient battery life.
- The mobile application will be run on a device with sufficient specifications: Android 4.4+, ios 9+, Chrome or Firefox web browser. These devices should support geolocation.
- The system will only work given sufficient reserve data.
- Users should have sufficient knowledge of using device on which the front-end software will be run.
- Administrators should have sufficient knowledge about databases and general console-based commands to run and monitor the server.
- Pilots should have sufficient knowledge of drone capabilities
- The system at any given time will only work for one reserve.
- The server hardware should be sufficiently capable of supporting multiple clients and performing necessary functions. Minimum recommended system specifications are: 4GB RAM, 4-core CPU.

9 Quality Requirements

9.1 Performance

QR1.1: The mobile application should start up in under 8 seconds on a mid-range device. This will be measured using a Samsung Galaxy S5.

QR1.2: The mobile application should maintain a frame rate of 30fps.This can be done using device system software or tools.

QR1.3: The system will generate a route within n seconds.This will be measured by logging the time the generate function was called and the time it ended.

9.2 Availability

QR2.1: Core functionality of the front-end application will be usable even if not connected to the internet, as long as the data has been pre-downloaded. This will be accomplished using service workers and will be tested using Google Chrome offline network mode.

9.3 Reliability

QR3.1: The system will back up the database hourly to a remote server. This will be monitored by viewing postgres backup logs on heroku.

QR3.2: The system will send crash reports to the client if any crashes occur. This will be accomplished by handling 404 and 501 HTTP responses client side and using timeouts

QR3.3: When the server is offline, the device will utilise caching to provide offline functionality. This will be tested using aeroplane mode on-device, or offline mode in browser.

9.4 Usability

QR4.1: The system will be streamlined and will require less than 5 taps to get to the desired view.

QR4.2: The system will be mobile-usable, tested by a score of at least 80 on Google Lighthouse.

QR4.3: The system will use Google Chrome Dev Tools Accessible Colours feature to determine whether colours are accessible. All colour combinations will be either AA or better rating.

9.5 Security

QR5.1: The system will hash and salt user passwords. This will be accomplished using SHA-256 and tested through unit testing.

QR5.2: The system will use HTTPS SSL encryption for all inter-device communication.

9.6 Scalability

QR6.1: The system will support a wide range of database management systems. This will be accomplished using ORM.

QR6.2: The System will support multiple Rangers and pilots at the same time. This will be accomplished using a connection pool to the server.

QR6.3: The system server will be capable of working on any server that can run Node.js and support the system's dependencies.

QR6.4: We do not currently have real-world reserve data, but the database should be chosen bearing the scale of this data in mind.

9.7 Flexibility

QR7.1: The system will be feasible to technological upgrades and updates. This will be accomplished using version control.

QR7.2: The system authentication module will be feasible to authentication protocol upgrades. This will be accomplished through modular system design.

9.8 Testability

QR8: All services offered by the system will be testable through unit tests and integration tests:
Unit tests will test whether the system functions and classes yield correct, predictable results.
Integration tests should test that all needed services from other subsystems are available and work together.
Acceptance testing will be used to determine whether the system meets the use cases and functional requirements described in this document.

The integration tests for each subsystem should test whether all services it needs from other subsystems are available, if the services are not available from the other subsystems, their services should be mocked.

10 Trace-ability Matrix Functional Requirements

	Map	Hotspot identification	Animal predictions	Route generation	Data modification	Geolocation	Notification	Authentication	Reporting
R1.1.1	x			x					
R1.1.2	x			x					
R1.2.1	x				x	x			
R1.2.2	x					x			
R1.2.3	x					x			
R1.3	x	x	x		x				
R2.1		x	x	x		x	x		x
R2.2		x	x	x		x	x		x
R3.1						x			x
R3.2.1						x			x
R3.2.2						x	x		x
R4.1	x				x		x		x
R4.2	x				x		x		x
R5.1							x		
R5.2							x		
R5.3							x		
R5.4							x		
R6.1				x					x
R6.2				x					x
R6.3			x		x				x
R7.1	x	x	x						x
R7.2	x			x					x
R7.3	x			x					x
R7.4			x		x				x
R8.1.1								x	x
R8.1.2							x	x	
R8.1.3								x	
R8.1.4							x	x	x
R8.1.5								x	
R8.2								x	x
R8.3.1					x			x	x
R8.3.2					x			x	x
R8.3.3					x			x	x
R8.3.4					x			x	
R9.1							x	x	x
R9.2	x				x				x
R9.3				x					x
R9.4	x	x			x				x
R9.5							x		x

Figure 4: System trace-ability matrix (functional requirements)

11 Trace-ability Matrix Quality requirements

	Map	Hotspot identification	Animal predictions	Route generation	Data modification	Geolocation	Notification	Authentication	Reporting
QR1.1	x					x		x	
QR1.2	x								
QR1.3	x			x		x	x		x
QR2.1	x				x			x	x
QR3.1					x		x		x
QR3.2							x		x
QR3.3								x	x
QR4.1	x				x				
QR4.2	x				x				
QR4.3	x				x				x
QR5.1					x			x	
QR5.2	x	x	x	x	x	x	x	x	x
QR6.1					x				x
QR6.2		x	x	x	x		x	x	x
QR6.3		x	x	x	x		x	x	x
QR6.4					x				
QR7.1	x	x	x	x	x	x	x	x	x
QR7.2					x			x	
QR8	x	x	x	x	x	x	x	x	x

Figure 5: System trace-ability matrix (quality requirements)