Final Project Report

A Prototype Rainforest Information System: to reduce the cost and increase the accessibility of technology to improve the collection of field data from tropical rainforest environments.

Student : Mark Anthony Start

Degree : BSc Computing and Information Systems

Supervisor :

Student Registration Number :

Submitted as part of the requirements for the award of the Degree in Bsc Computing and Information Systems of the University of London International Programmes.

"Over half of the world's plants and animal species are found in the rainforests,
99% of them have still to be studied by science."
- Eden Project, Rainforest Biome.

Table of Contents

Abstract / Summary

Chapter 1 Terms of Reference

1.0 Introduction
1.1 Project Aims
1.2 Objectives
1.3 Deliverables
1.4 Justification
1.5 Deviations from the Preliminary Project Report (PPR)

Chapter 2 Literary Review and Methods

2.0 Literary Review Introduction
2.1 Xprize Rainforest
2.2 The Smithsonian Tropical Research Institute Research Project
2.3 The Norwegian Polar Institute Research Article
2.4 Wildlife ACT
2.5 SMART Collars Research Project
2.6 Rainforest Connection (RFCX) Internship
2.7 Global Forest Watch
2.8 Regnskogtondet Rainforest Foundation Norway (RFN) Project Guide
2.9 International Tropical Timber Council (ITTO) Summary Report
2.10 Open Forests
2.11 WildLabs
2.12 Humanitarian Nomad
2.13 Software Application Comparison
2.14 Smart Devices
2.15 Literary Review Conclusion

2.16 Methods Introduction
2.17 Project Methodology
2.18 Additional Project Methodology Concepts and Implementation Methods
2.19 Project Plan
2.20 Additional Research and Interviews
2.21 Rainforest Information System Task Analysis and Scenarios
2.22 Requirements Specification
2.23 Test Cases

Chapter 3 The Design, Development and Implementation of the Prototype

3.0 Introduction
3.1 System Architecture
3.2 Overview of the Proposed System - Rough Sketch
3.3 Track and Point Data Model
3.4 GPS Precision
3.5 Input Component
3.5.0 Input Component Design

3.5.1 Refined ‘Input-Data Use Case’
3.5.2 Noun Phrases and Class Identification
3.5.3 Class Diagram
3.5.4 Activity Diagram
3.5.5 User Interface
3.5.6 Hardware
3.5.7 Input Component Implementation
3.5.8 Iteration One
3.5.8.0 Analysis Stage
3.5.8.1 Design Stage
3.5.8.2 Implementation Stage
3.5.8.3 Testing and Iteration Retrospective
3.5.9 Iteration Two
3.6 Central Processing and Data Storage Component
3.6.0 Central Processing and Data Storage Component Design
3.6.1 Server
3.6.1.0 Server Implementation
3.6.1.1 Server Configuration
3.6.1.2 Server Application Code
3.6.1.3 Server Unit and Integration Testing
3.6.2 Database
3.6.2.0 Relational Database Design
3.6.2.1 Modelling for the Conceptual E/R Diagram
3.6.2.2 Refining the E/R Diagram
3.6.2.3 Transforming the E/R Diagram to a Relational Model
3.6.2.4 Normalisation to BCNF 3.5 Normal Form
3.6.2.5 Koppen Climate Classification Types
3.6.2.6 Database Implementation
3.6.2.7 Database Testing and Retrospective
3.7 Output Presentation Component
3.7.0 Web GL Earth
3.7.1 Analysis, Design and Retrospective Stages
3.7.2 Implementation
3.7.3 Web Page Summary
3.8 HTTP, TCP and Satellite Communication
3.9 Prototype RIS Hardware

Chapter 4 Testing

4.0 Input Component
4.1 Central Processing and Data Storage Component
4.2 Output Presentation Component
4.3 Usability Testing and User Interviews
4.4 Field Trials of Data Retrieval and Accuracy in Tropical Rainforest Conditions
4.4.0 Field Test Experiment 1
4.4.1 Field Test Experiment 2
4.5 Lab Testing

Chapter 5 Results of Use

5.0 Input Component Video Walkthrough

5.1 Processing and Data Storage Component Video Walkthrough
5.2 Output Component 3D Globe Webpage Video Walkthrough
5.3 Results of System Testing

Chapter 6 Discussion of Results

6.0 Discussion of Test Results
6.1 Discussion of Usability, User Acceptance and User Interviews
6.2 Discussion of Software and Project Methodology
6.3 The Cost and Accessibility of the Prototype System

Chapter 7 Conclusions

7.0 Current State of the Project
7.1 Current Limitations

Evaluation

Appendix A: Source code

A.0 Input Component
A.1 Central Processing and Data Storage Component
A.2 Output Presentation Component

Appendix B: List of Search Terms and Literary Review Links
Appendix C: References, Figures and Tables

Abbreviations

RIS	Rainforest Information System
FIS	Forestry Information System
IS	Information System
GPS	Global Positioning System
FPR	Final project report
PPR	Preliminary project report
TPS	Transaction processing system

Abstract / Summary

A Bsc Computing and Information Systems project to detail the current innovations and to attempt to improve the current limitations of tropical Rainforest Information Systems (RIS) including reducing their cost and increasing the accessibility of technology for tropical rainforest scientific field research, forestry, data collection and visualisation of field recorded sensor readings.

…

Rainforests can be inaccessible, their mystery, extensive biodiversity of species, and extreme conditions make rainforests areas of great, and challenging, scientific research. Technologies used in the rainforests have to be designed to span the vastness of these forests, record their complexities and resist and perform in intense tropical rainforest conditions.

From research into the current state and context of innovations and systems that meet these core requirements, and comparing the performance, cost and accessibility of the existing systems, their software, hardware, and recognizing that most of the world's tropical rainforests are found in developing countries, it was determined that recycled devices such as smartphones and tablets, due to their prevalence, ubiquity, mobility, familiarity, and their conveniently builtin array of environmental sensors, networking hardware, protective casing and being well tested in a range of operating conditions could be a viable, low cost, accessible, easy to deploy and use, alternative to the high cost, complex, high precision, high barrier of entry, well funded scientific and Forestry Information System (FIS) technologies that exist with their own strengths and limitations.

The project aims to measure the feasibility of using recycled hardware and open source software as part of a low cost, easy to use, robust prototype RIS to retrieve scientifically useful information easily and at a low cost when conducting field research in the world's tropical rainforests and in tropical rainforest conditions.

The project report documents the research, design, development, software code, testing, field trials, evaluation and user feedback for the feasibility, cost, accuracy and availability of using open source software and recycled hardware in the prototype RIS. The complete prototype is made with a 2013 Samsung Galaxy S4 smartphone and a 2012 Google Nexus 7 tablet, a 15w solar panel and a 9000 mAh power bank.

A client smartphone Java Android application serves as a reasonably reliable automated data-capturing TPS (Transaction Processing System) Input Component of the RIS, that automates the data collection from the currently implemented device's inbuilt sensors, including, GPS, temperature, humidity and timestamps at 15 minute intervals that can store the field data offline and network data to the Central processing and Data storage component; a Java Springboot web server and Docker based Postgres database using a GSM or wireless Internet connection.

Retrieved data hosted from the server is then visualizable on the Output Component's presentation dashboard web page by mapping the GPS coordinates and sensor readings on an interactive 3D globe with customizable map overlayers including satellite and road.

The complete system is durable and portable and can be taken into the field with the server running on a full desktop distribution of Linux running on a modified 2012 Google Nexus 7 tablet as an 'in-the-field laptop' hosting the server and database and making available the stored field data with a HTTP RESTful API.

The system can be powered by a 15w solar panel and power bank and with the right management of its operation and power reserves can be powered without interruption.

Accuracy of the input component's sensors were tested and evaluated and shown to be reasonable for the humidity sensor and quite accurate for the temperature sensor in comparison to online weather services and a commercially available device with the same set of inbuilt sensors. GPS coordinates ranged from exact to within a variance of ~50-100 metres of their true location with reliable timestamps but with some anomalous time and date values.

The overall cost of the prototype was very low and the accessibility of the system evaluated to be high in comparison to the technologies of the researched scientific, academic and commercially available systems.

The feasibility of the performance of using recycled devices was also found to be high but necessitates adaptable, modular and extendible design and engineering principles, project methodology, system architecture and an iterative development cycle based on testing results and user feedback to enable continuous development to more closely align with the identified primary and secondary requirements of the system formalised from the literary review and to improve the systems performance and accuracy of its components based on user feedback and testing results.

Screenshots of the complete system are provided below.

3 video walkthroughs of the system are available from
https://www.youtube.com/user/markstart00

1. https://www.youtube.com/watch?v=Cb32cDj3JFc
2. https://www.youtube.com/watch?v=kujqOj48kVg&t
3. https://www.youtube.com/watch?v=0TvVNaYX_e0&t

The project Github pages:

https://github.com/SatellitesAndRainforests/Rainforest-Information-System-Documentation
https://github.com/SatellitesAndRainforests/RainforestApp
https://github.com/SatellitesAndRainforests/RainforestServer
https://github.com/SatellitesAndRainforests/RainforestPostgresDatabase

Project's Keywords: Information Systems, Rainforests, Sensors, GPS, Humidity, Temperature, Android, Java, WebGL Earth, PostgreSQL, Linux, Prototype, Recycled Smart Devices.

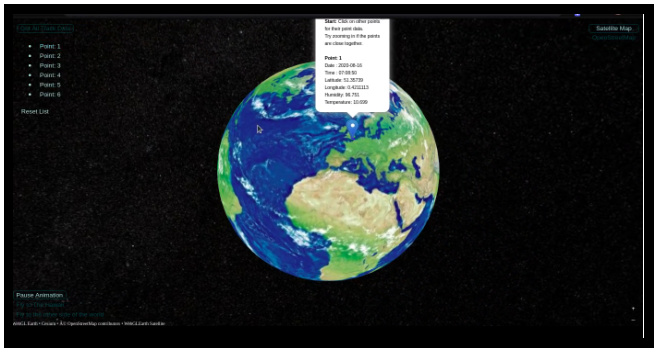
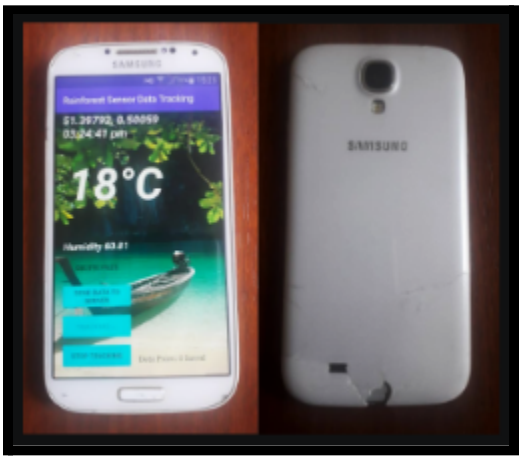


Fig. 1-5: The complete Rainforest Information System

Fig. 1: The Input Component Java Android client application
Fig. 2: The Central processing and data storage Component and recycled system hardware
Fig. 4: The Output presentation component 3D globe dashboard-webpage
Fig. 5: The Track and Point data model mapped on the interactive globe with system collected sensor readings for the GPS location.
Fig. 5: The Input Component recycled hardware



Chapter 1 Terms of Reference

1.0 Introduction

The rainforests of the world are of significant scientific, environmental and technical challenge and interest due to their harsh conditions, high precipitation, humidity, temperature, extreme wildlife, destructive insects, accelerated deforestation, illegal poaching and that most of the rainforests of the world are in developing countries. A perfect setting for the attempted innovation of an academic scientific research project. Rainforests are also of personal lifelong interest and pairing them with computers was ideal.

The subject of my project was chosen to both increase my understanding of where technology (my other lifelong interest) and rainforests intersect.

The choice to explore the feasibility of recycled technology concluded from the literary review and research, but during this project I completed an internship with Rainforest Connection (RFCX). A San Francisco based technology and environmental sciences company, which uses recycled smartphones to collect bioacoustic audio data from rainforests around the world, and uses the GSM network to retrieve data to a cloud server database. They currently have begun to apply artificial intelligence to analyse the large amounts of tropical rainforest data (my internship was to help with the analysis of the accuracy of the AI's identification of the audio signatures of illegal logging activity). The use of smartphones in RFCX's project are detailed in the literary review and closely correlate with my own reasoning and thinking.

The project was chosen to be able to research and showcase in a best light a technical and literary body of work to show respective employers in Rainforest Conservation, Information System development, software engineering, systems analysis, database design and development, Java programming, Android development, Javascript web development, server side development, client side data visualisation, computer networking and data communications, which, working in, are all areas and professional objectives of mine.

To make Information Systems a focus of the project was to reflect and make the work the most relevant to my degree program so that the project could formalise and condense my technical, and theoretical understanding of computing and my efforts in the individual subjects into one complete real-world Information System and project of work. This is designed to show my strengths and showcase my skills as well as improving my understanding of the individual Information System components and computing subjects, including computing project management and methodologies.

I expect my interest in developing rainforest technologies will continue and I already have some great new future project ideas !

Relevant courses

- Information Systems: foundations of e-business
- Introduction to Java and object-oriented programming
- Introduction to computing and the Internet
- Database systems
- Graphical object-oriented and internet programming in Java

- Data communications and enterprise networking
- Information systems management
- Software engineering, algorithm design and analysis
- Software engineering project management
- Interaction Design

1.1 Project Aims

The aims of the project are to find out how technologies, software, hardware and system architecture, have to be specially designed to be applied and used in tropical rainforest conditions.
How systems retrieve tropical rainforest field data and if it is technically and socio-economically feasible to use vendor free open source software and recycled devices as component hardware to reduce the cost, the barrier of entry, improve the flexibility, accessibility and adaptability of meeting new requirements, the ease of deployment and the use of RIS technology to improve the limited research completed in the world's tropical rainforests. And evaluate if this approach can be used as a viable alternative to current scientific and FIS technology, their cost, accessibility and their own offset balance of advantages and disadvantages and socio technical limitations.

Framed as a question "Why has the rainforest, similar to the ocean remained unexplored and under researched, where is technology being used, and how can its cost be reduced, and its use increased, to reduce the current limitations of tropical rainforest research and data collection?"

A more refined aim is to :

Research, design, implement, test, evaluate and field trial a prototype RIS that makes use of recycled devices and open source software to determine their feasibility to easily, accurately and automatically increase the collection of information from tropical rainforest environments.
The prototype should have an input TPS component, be able to networked data to a central processing and data storage component (server and database) and host that information to an output presentation component web page dashboard to clearly display the information retrieved for data analysis, system evaluation and to evaluate the prototype system's performance in comparison to current RIS and FIS technology including its cost and accessibility.

1.2 Objectives

To accomplish the aims and answer the above question I set out to accomplish the following objectives:

- Research and understand the current context of technologies used in tropical rainforests.
- From the research formalise a list of requirements needed in a prototype RIS.
- Formalise a list of requirements for the system architecture, software and hardware of a prototype that can meet the project's aims including the reduction of cost, the increase in accessibility of RIS technologies and possibly desirable system features or performance.

- With the above, design a low cost and easy to use Rainforest Information System that is practical for use in the rainforests of the world to collect rainforest data for data visualisation and analysis and to make the data available to people of rainforest communities.

- Design and Implement the components of a complete Information System aligned to the requirements that can network data from an input component in a rainforest, to a central processing and data storage component, where the information can be made available to an output component, reminiscent of a dashboard for data analysis, system performance feedback and overview.

- There is an objective that the system should enable a complete feedback loop for the Information System so that the system's performance and the quality of the data it collects can be improved.

- Test the system's components technically in isolation and as a complete system for reliability, accuracy, runtime errors and anomalies.

- Field trial the prototype system to evaluate its performance for use in real world tropical rainforest conditions with a focus on evaluating the retrieved data for its accuracy and precision for the set of implemented sensors and their recorded sensor readings.

- Design the system to be flexible, modular and extendible to first meet the primary requirements and then to be adaptable to user feedback and the implementation of more sensors and to increase the depth and breadth of the prototype's usefulness and implemented functionality.

- Conduct interviews to evaluate the reception of the system and user experience and analyse those observations to improve the system or initiate future iterations of design, development, feedback and evaluation.

- Evaluate the prototype systems potential to be used as a viable alternative to current systems.

Many of the objectives of the project are technical but with any Information System consideration for a cost-performance balance and socio-economic factors are important particularly to understand how to reduce the cost and increase the accessibility of the system to be able to improve rainforest research and to be suitable for use in developing countries where the majority of the world's tropical rainforests are found.

1.3 Deliverables

- Final project report

- Preliminary project report

- An academic and other resources literature review on the current state and innovations in the field of tropical rainforest technologies including RISs and FISs for tropical rainforest data collection. Researching sources, approaches, methods and technologies appropriate for use in this project.

- Formalised primary and secondary requirements for the system architecture, software, hardware, functional and non-functional aspects and the data model for the prototype RIS.

- An overview design of the complete Information System architecture and the technical design of the Input, Processing and Data storage and Output components of the prototype system, including UML, Activity and Class diagrams to clarify important functionality.
- Database conceptual and E/R diagrams and a normalised SQL schema.
- An Input Component Java Android client application.
- A central processing and data storage Java Springboot web HTTP RESTful API server and Docker based Postgres database.
- An Output presentation Component interactive Javascript 3D globe data visualisation web page.
- Lowcost rainforest specific recycled smartphone and tablet hardware designed field computers with additional field solar and battery pack power sources.
- Results of testing and field trials of the prototype RIS.
- Discussion on the evaluation, limitations, strengths, improvements and future work for the system including the prototypes comparison with conterparay researched RIS systems.
- Video walkthroughs.
- Github project links.

1.4 Justification

The conducted literary review research provides a starting point and context for the understanding of the current field of scientific and forestry industry application of technology in the field of tropical rainforest data collection, measurement, monitoring and data visualisation, as well as highlighting current systems, costs, accessibility, strengths and limitations.

Exploring the use of open source and recycled hardware aims to incorporate the strengths of the ubiquity, convenience, inbuilt sensors, tested and protective casing, design for use in a range of geographical environments and the mobility of smart devices and the potential to substantially lower the cost of the prototype system and to see if this approach and these strengths can be more effective or increase the collection of unsurveyed areas of the world's expansive tropical rainforests.

The software architecture of the system follows a client-server model so that data can be centralised and many smartphone clients can contribute data to the database as rainforests are located in a band between the Tropics of Cancer and Capricorn around the globe.

A software engineering approach to design, implementation, testing and evaluation are intended to give the project the advantages of a successful engineering project structure to increase the value, desirability and usability to users of the prototype RIS so that they may find the system useful and usable and so it can be further developed to align to user feedback and test results and more closely aligned to identified system requirements and user needs.

1.5 Deviations from the Preliminary Project Report (PPR)

The focus of the FPR differs from the focus of the PPR. The PPR quite accurately describes the aims, objectives, deliverables, methods, designs, implementation and testing phases of the project although in a much more general way.

Based on the feedback I received, the focus of the FPR changed in addition to the original aim; to understand technologies used on tropical rainforests and attempt to design and implement a prototype that could be an improvement in some way to collect data from tropical rainforests.

The new focus included focusing on improving the price and accessibility of the prototype in addition to its ability to more effectively collect data from tropical rainforest environments when compared to current academic, scientific and commercially available systems so that the prototype and project could have additional measurable benchmarks to evaluate if the prototype and the project met its original aims.

Some extracts from the PPR and included in the FPR as they remained very accurate in describing different aspects of the project, particularly the project methodology, additional methods used to conduct the project and its architectural design.

Chapter 2 Literature Review and Methods

2.0 Literary Review Introduction

The rainforest is a unique environment that requires unique considerations for technologies to be effective. The literary review attempts to highlight the strengths, weaknesses, costs and accessibility of existing projects, RIS, FIS, software, hardware and approaches to the complexities of designing, implementing and operating technologies in tropical rainforests to gather data and information.

The literary review highlights 'project keys' from different sources for this project to explore for the design, implementation and deployment of a lower cost, more accessible and possibly improved performing prototype RIS.

Relevant sources were researched for the design, methods, implementation, architecture, software and hardware, and project methodologies, that could help this project to meet its aims and objectives focusing on the design and development of a prototype RIS that tests the viability of using open source software and recycled hardware and attempts to exhibit improvements when compared to existing systems.

The literary review gives an outline of the current context of the field of rainforest technologies used in biodiversity monitoring, rainforest and tropical Forestry Information Systems, illegal logging and poaching and detection, animal tracking and field data collecting hardware and software applications. It includes research papers, articles, organisations websites, a comparison of existing software and hardware and information from other sources and serves as the starting point for the project.

Researching and analysing different approaches and technologies from useful and insightful sources will enable the project to identify and understand the core needs and requirements for the operational and environmental considerations for the prototype system that are useful, effective and desirable for users.

The literary review was formerly conducted in the first 3 weeks of August 2020, but informally also in the spring and summer of 2020.

Google Scholar was used to retrieve sources of academic quality that could be reliably sourced, including peer reviewed journals, research projects and papers, but did not exclude valuable analysis and research from less formal sources including organisations websites, communities, blogs and articles that provided valuable qualitative insight into understanding the current context of the field.
Recent projects and papers were sought but sources were not precluded based on publication date if the information was insightful.

A list of search terms are included in Appendix B.

Main points relevant to the project are summarised in small 'project key' summary sections at the end of each reviewed source.

2.1 Xprize Rainforest

www.xprize.org/prizes/rainforest
www.youtube.com/watch?v=mEYHUEGNgXk&ab_channel=XPRIZE

Xprize has launched a five year long, Harrison Ford fronted, $10,000,000 prize competition to promote and accelerate the automation and innovation of how to rapidly catalogue the vast and complex biodiversity of the world's tropical rainforests.

The technical challenge of the data capture of the extraordinarily large scale of the rainforest's expansive geographical range and it's almost exponential complexity of biodiversity is a considerable challenge. From personal experience the rainforest can appear to go on forever and is very difficult to traverse and navigate.

The relatively high monetary incentive, technical challenge and celebrity of Indiana Jones, highlights the recognised potential for innovations and rewards to be made in rainforest technologies, and although, has not gained much attention, captures the setting of the current context of research in the field at this time.

The link is included above for a recommended ~2 minute introduction to the organisation's competition and context of the field.

The problem of how to automate the survey of tropical rainforest data collection is a key objective of this project's prototype.

Project Key:

- How to collect all sorts of data, quickly, from very large tricky tropical rainforest environments.

2.2 The Smithsonian Tropical Research Institute Research Project

Tracking Animal Location and Activity with an Automated Radio Telemetry System in a Tropical Rainforest

The research project and paper of the Panamanian island-wide information retrieval and animal tracking network, details and reports on the use of strategically placed radio telemetry antenna transmission towers that transmit radio signals to locate radio receiver tagged species within the range of the Automated Radio-Telemetry System (ARTS), collecting data of “science quality … [and of] higher resolution” in a tropical rainforest environment over existing systems (including GPS systems) that can monitor a variety of different sized rainforest animal species (Kays, 2011, p.2).

The project's strengths are in the increased frequency of the samples and the accuracy of the data captured when compared to other technological approaches such as the very popular but expensive remote animal and weather tracking satellite system (ARGOS) and existing traditional field biologist survey methods (explored later).

The comparison below highlights the ARTS system’s advantage when compared to traditional radio telemetry, GPS satellite tracking and ARGOS satellite tracking. The ARTS system can automate the data collection for the smallest animals of the compared systems and with the lowest effects of interference from high tree canopies and considerable tropical rainforest vegetation.

Tracking method	Smallest tags, animal weight	Automated data collection?	Interference by vegetation
Traditional radio telemetry	0.2 g, 4 g	No	Low
GPS satellite tracking	10 g, 400 g	Yes	Medium
Satellite tracking (ARGOS)	10 g, 200 g	Yes	High
Automated radio telemetry (ARTS)	0.2 g, 4 g	Yes	Low

Fig. 6: Comparison of tracking methods (Kays, 2011, p.3).

Disadvantages of the system include the necessity to tag animals with invasive and limiting hardware[1] that can affect their behaviour, cannot monitor all sizes and types of animals, increases the cost and time to establish the system, introduces a technological effect and the potential to pollute the environment, and, importantly, is only operational as long as the animals are in range of the radio antenna towers (Kays, 2011), but can work very well on an island or where there is is existing utilisable infrastructure.

The project highlights a significant advantage in the cost and accuracy within a fixed limited range of radio telemetry over expensive satellite systems and details important design considerations when implementing and operating technology in tropical rainforest conditions.
An interesting observation is the importance of the positioning of the antennas in the rainforest canopy relative to the forest floor that has a measurable effect on the system's performance.
Some of the interesting physical considerations for deploying technology to the rainforest included "Large tree fall on [tower] … two total times" and "lightning strike[s]" … (Kays, 2011, p.13), perhaps an overlooked natural event in an environment with naturally very high levels of precipitation.

The ARTS system is a successful innovation in the automation of tropical rainforest research considering its accuracy, sample rate and cost but is limited by its range, infrastructure and relative operational complexity and the disruptive exposure to the effects of the natural environment.

Key technologies used in the project include :

- Radio-telemetry 40m receiver towers strategically placed around the island.
- Radio animal antenna transmitters to monitor and locate different species.
- A Web-accessible database for the centralisation and ease of access to the retrieved data for analysis and visualisation.

Project Keys:

- Automated data collection is more desirable than traditional survey methods.
- A gain in one direction, can be a loss in another. Smaller scale, higher resolution data can be more insightful scientifically in one way where more disrupted, larger scale, satellite data can be valuable in another.
- Very high initial and running costs with very low accessibility outside of the academic community.
- Automated retrieval of data, database storage and web-based accessible interface for the data.
- The rainforest weather, events, canopy and vegetation can cause interference to the collection and transmission of the collected data.

[1] Animal tracking devices can be a maximum of 4% to 5% of the animal's body weight (wildlifeact.com, 2019).

2.3 The Norwegian Polar Institute Research Article

Arctic fox dispersal from Svalbard to Canada: One female's long run across sea ice

A research paper on the exceptional distance covered by an arctic fox tracked with a satellite collar and satellite telemetry by the ARGOS system over the extraordinary distance from one ecosystem in Norway to another in Canada over 2000km away is a great example of the advantages of a satellite biodiversity tracking system (although almost as far from equatorial tropics as possible) when compared to the tropical rainforest radio tower telemetry system of the ARTS project. This source is included, as the research paper and the map included below were a great inspiration for this project.

ARGOS has been a key technology in wildlife tracking since 1978.
The use of satellites to network data can be seen as a radio-relay that routes data from one location on earth to another different location on Earth.
Satellites tracking is affected by the doppler effect; a shift in the frequency of a radio wave between a relative source and moving destination so that its accuracy is limited to about, "within 150 metres" (argos-system.org, 2019).

The research paper illustrates the practical application of the ARGOS system to retrieve less accurate, disrupted, information from very useful long range tracking that can reveal comparatively very interesting information.
The map illustrates the type and quality of the data that can be collected.

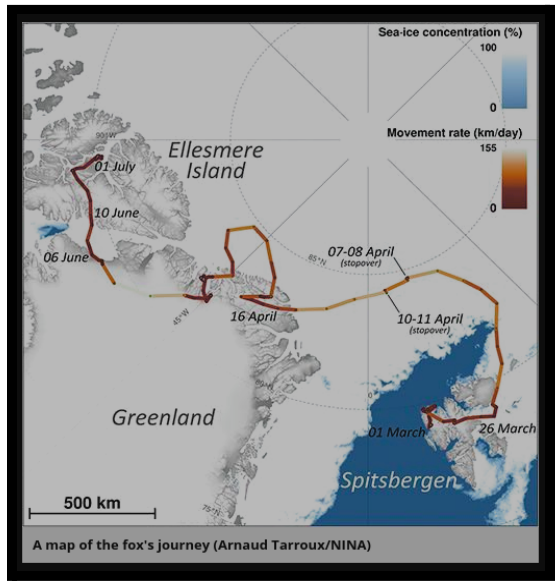


Fig. 7: A map of the journey (Fuglei & Tarroux, 2019, p.3).

The paper details how data can be filtered to "remove…[the] positions for which the location process had failed" (Polar Research, 2019, p.2) and further filtered to visualise the data for a selected insight or effect by making use of programming libraries (bsam) and functions (fit_ssm) to refine and process the retrieved data. The paper includes very informative visualisation and displays a very useful function of a system; filtering and projection of data onto a map, that adds insight and value from the perspective of the end users of the system for data analysis and when reviewing a systems performance.

Project Keys:

- Satellite use is expensive[1] due to the limit of networking capacity, high initial cost and risk.
- Their application is well established and structured and can deliver excellent although less accurate large surface area data.
- The ability to filter anomalies and further refine data to reveal insightful information adds considerable value to the system.
- The ability to plot GPS points and other retrieved information onto a map for visualisation and analysis is a considerably valuable function of the system.
- The application and accessibility of the ARGOS-SYSTEM is typically project specific.

[1] ARGOS animal GPS collars cost ~$2,000 to ~$5,000, with a service fee of ~$500 "nearly 10 times ... a VHF tag" (Sue Palminteri, Mangabay.com, 2017).

2.4 Wildlife ACT

GPS and VHF Tracking Collars for Wildlife Monitoring Comparison

Different tracking collars and transmitters have different strengths and weaknesses. Wildlife ACT a team of experienced wildlife field conservationists compare and evaluate different tracking collars for use in their long term, sustainable African game reserve animal monitoring and management of South African animals including rhinos, cheetahs, leopards, lions and elephants.
A summary of their professional comparison is included as valuable conservationists' qualitative insight and a brief summary of available commercial technologies to retrieve varied information for research and animal behaviour monitoring and management from different habitats, environments and for a range of different animals.

In summary :

VHF Very High Frequency pulse collars emit a radio wave that can be observed by a receiver and an antenna. They are reliable, affordable, have a long lasting battery but with a shorter range.

GPS Global Positioning System collars log coordinates in real-time by using triangulation and satellite signals. Satellite collars can upload the log to another satellite or to another network that can be used to communicate data, such as, WiFi, GSM and antennas. With shorter battery life, more cpu intensive programs, heavier hardware and a more expensive cost. The GPS collars evaluated were indicated to have an accuracy of ~5 metres, much more accurate than the ARGOS system mentioned above.

Collar Type	Additional Information	Battery Capacity	Cost
VHF	More field work intensive. Can capture behavioural data not retrievable from other collars. Can be incorporated with GPS collars.	Varies, up to 3 years.	~$200 Least expensive
GPS/ GSM	Automated GPS and data capture with the GSM or local network. Includes Google Earth mapping and visualisation	Varies, up to 2 years.	~$500 to ~$2000
GPS Satellite	Automated GPS and data capture through satellite email protocol. Includes Google Earth mapping and visualisation. Excellent for inaccessible areas.	Varies, up to 2 years.	~$2500 Most expensive

Table 1: Comparison of tracking collars

Additional sensors, and customizable battery capacities can be incorporated that can retrieve additional and different types of data, for longer durations, depending on the animal being tracked, their ranges, habitats and environments.

Project Keys:

- GPS systems exist that are more accurate than ARGOS.

- A software, hardware, cost, complexity balance exists with the balance of battery performance, the type of information, accuracy and the volume of data retrievable.
- GoogleEarth mapping and visualisation for satellite coordinates, animal and environment information.
- Additional sensors can capture more comprehensive data when less cpu intensive software and hardware is used.

2.5 SMART Collars Research Paper

Instantaneous Energetics of Puma Kills Reveal Advantage of Field Sneak Attacks

This research paper details the innovative research project and invention of the SMART GPS animal tracking collar, an analysis and review of its performance and a very detailed quantitative data analysis of the very high frequency of data samples and large volume of datasets that it is capable of capturing.
Invented by eco-physiologist Terrie Williams, wildlife ecologist Christopher Wilmers, and computer engineer Gabriel Elkaim of the University of California, SMART collars incorporate more sensors than existing GPS collars, that can collect more, and more precise information, including 3D movement of animals, up and down trees, elevation, activity in the canopy and forest floor, and with a much greater sample frequency (32 times per second), with the capability of recording jumps, acceleration, and an animals speed of motion.
The design of the collar and sensors can collect very insightful 'data-signatures' that can be used to derive the energy expenditure of animals, where and whether they are sleeping, eating and hunting.
With the advantage of the increased frequency and comprehensive complexity of the data points is the challenge of analysis of the greatly increased size of the data sets collected, requiring considerable computer storage, man hours and computational power to process.
Due to the necessary size of the transmission of the large volumes of data through a GSM or satellite network, networking data is not used in favour of retrieving the collar and data from the tracked animal in the forest.

> "You can't help wanting to preserve them when you know the intimate details of their lives."
> —Terrie Williams.

Project Keys:

- More sensors at higher sampling frequencies can capture very precise and intimate animal and environmental information.
- When retrieving large data-sets, there is a cost-practicality balance between networking and retrieving the data from the field by hand.
- Not available commercially, cost: n/a, accessibility: n/a.

2.6 Rainforest Connection (RFCX) Internship

Internship, and the Prevention of Illegal Logging and Biodiversity Monitoring with Real-Time Alerts, Audio Live Streams and Bioacoustic Artificial Intelligence (AI) and Big Data Analysis with Cloud and GSM Networking

On the 9th of August, 2020 I contacted the San Francisco based Rainforest Connection not for profit tech organisation at their website by email asking about collaborating with their project and later in 2020 I applied and was given a place to intern and help with the analysis of the performance of their third party

Huawei artificial intelligence plugin to detect different types of audio signatures in their recorded audio rainforest cloud data, and, got a inside sneak peek at their impressive web based rainforest Information System!

Rainforest Connection has developed a rainforest monitoring and protection system that responds with real-time alerts for the audio signature detection of illegal logging, deforestation and poaching identification, such as the sounds of chainsaws, trucks and gunshots.
"Upcycled smartphones" with their built in and optionally attached additional sensors, including powerful microphones (techcrunch.com, 2014) are used to record and detect the audio signature of trucks, illegal logging equipment and for the identification of different species of birds, insects and terrestrial animals.
The modified smartphones then use the existing GSM cellular network to network the data to the cloud where SMS notifications with GPS coordinates and other information can be sent to partner organisations such as wildlife conservation rangers that can react to intercept and stop the illegal logging activity at the geotagged location.
Solar panels and battery packs are used to be able to power the modified smartphones for several years and are strategically and covertly installed at locations around rainforest preserves, in the rainforest canopy and at different national forest locations.

RFCX's technology collects the biggest shareable, searchable 'Audio Ark' of rainforest sounds and eco-data from tropical and subtropical rainforests in the world (rfcx.com, 2019). In comparison with other rainforest surveillance systems such as satellite imagery where updates rely on satellite orbits, information can be infrequent "you can only know about logging activity days or weeks after it has occurred" (Topher White, Youtube, 2013).

Different RFCX smartphone applications can live stream the retrieved audio feeds, map the location of the 'Rainforest Guardian' devices and receive GSM alerts from the Guardian installation locations in different rainforests from around the world. Their website is well worth the visit.

From the internship I learned the practicality of networking high volumes of live streaming audio data through the GSM network using SMS, how a complete web accessible RIS is designed and implemented and how its data becomes accessible globally, and how retrieved rainforest data can be visualised, analysed and reviewed in reports, interactive maps and with interactive multimedia plugins to improve the RIS's accuracy and performance.

Project Keys:

- Recycled hardware and smartphones can considerably save costs, are ubiquitous around the world, can be easily sourced and can viably serve as components in a RIS when modified to operate in semi-remote tropical (hot and wet) conditions by including solar panels, battery packs and insect proof casing.
- Smartphones already have many built in environmental sensors and incorporated networking technologies including GPS, GSM, temperature and time, and many are Linux based.
- Large volumes of data can feasibly be retrieved from tropical rainforest conditions and environments with existing GSM network infrastructures with considerable advantages over satellite systems or by hand from the field, including closer to real time information, saving time and cost.
- Web based RIS's interfaces can easily and conveniently visualise and retrieve rainforest data from a database to be used for data analysis and to evaluate a systems performance and accuracy.
- Low commercial availability, requires inhouse installation, costs vary across projects but in general are high.

2.7 Global Forest Watch

Open Source Web-Based Satellite Forestry and Deforestation Tracking and Visualisation

Global Forest Watch is an excellent web-based visualisation of satellite imagery and an analysis tool for monitoring forest changes and events. A world map of the world's forests sourced from a compilation of hundreds of forest data-sets that can be further customizable with additional data that can be used to build custom maps and visualise a new project's information.
With areas of near real-time forest overview achieved with the additional upload functionality for deforestation, forest fires and other event information from on-location smartphone applications and other data entry sources.
The Global Forest Watch ForestWatcher application can collect field data, send data, and send and receive alerts similar to the Rainforest Connection system to locate illegal logging activity and other events. It is not automated and can only partially work offline without access to the cloud database.
Global Forest Watch's web-based interface serves as an excellent public facing dashboard, unlike Rainforest Connection that is privately facing, similar to that of an Executive Information Systems used for data analysis and making long term strategic decisions based on retrieved information from other components of an Information System, in this case, satellite and mobile device applications similar to input TPS data collecting components.

Global Forest Watch extends and incorporates proprietary ArcGIS software, an excellent mapping tool, to visualise the location based data requiring private licensing.

Project Keys:

- Web-accessible and interactive global view of data similar to an Executive Information System dashboard, including zoom levels, different map overlays (satellite and terrain) and selectable data provides excellent data customisation and presentation.
- Open source vs proprietary mapping softwares.
- The customization functionality of the presentation of collected data.
- Smartphone applications for manual data-field entry that can store data locally and work offline.
- Complete Information System structure of input/retrieval, processing/compilation and output/presentation components.
- Availability and accessibility is very high and access to the dashboard overview forestry information is global and free.

2.8 Regnskogtondet Rainforest Foundation Norway (RFN) Project Guide

Using Technology to Monitor and Share Information on Rainforests and Forest Peoples' Rights

A leading international organisation for the protection of rainforests and indigenous rainforest people, Regnskogtondet Rainforest Foundation Norway collaborated with 'The Engine Room' to release a guide of how to successfully use and apply technology to share rainforest information and protect the rights of native people from the review of collected operations and projects.

The guide has been incredibly useful for this project as its very aim is to outline and highlight points for the successful design and completion of rainforest technology projects. Designed as a starting point, the whole guide was useful but without including every part of the guide that I found helpful, a summary of main points and the recurring themes for success are outlined:

Toward the beginning of the guide a useful observations is made that most successful technological rainforest projects are :

- Solidly designed
- Have clear goals
- And have an understanding of the field, the problem at hand, and the technology at hand.

"A successful project requires explicit goals and a solid strategy"
(Rainforest Foundation Norway, 2016, p.9).

The guide highlights strategies and case studies of successful technology projects operating in the rainforest as well as how to organise, manage and use the data/ information collected, guides on different types of technologies, tools, and how to successfully apply them, including:

Smartphones, satellite imagery, GPS, drones, audio, video, maps, GIS, and their application to forestry, conservation, poaching, logging, scientific research and the protection of native peoples.

The guide promotes and highlights the recent trending improvements that have made technology more useful in rainforests, including:

- Internet network and access
- The telecommunication networks expansion into remote areas
- Ubiquitous smartphones availability and applications
- Recent hardware and software performance improvements and decreases in cost.
- The availability of satellite imagery.

Identifying these as key tools that have made impactful improvements to achieve collecting, retrieving, and sharing rainforest data and information.

Project Keys:

- ***Smartphones***
  - Smartphones and tablets are highlighted as particularly useful as small inexpensive available powerful computers with reasonable processors, storage, many inbuilt sensors that can collect and transmit field data directly with inbuilt networking channels.
  - Software can run easily on both low and high end devices and make use of low end services such as SMS (Short Message Service) and high end device sensors such as GPS, and high resolution cameras.

    The guide identifies that design specific technologies (e.g GPS trackers) are more precise, function more consistently and are more durable in rainforest environments than mobile devices.
  - Can automate, validate and transmit to a database the time-consuming form-data processing of hand-written form data collection particularly in wet and distant remote locations and when transcribing for data entry.

    Although in some ways technology can be less resistant to a tropical environment particularly in high precipitation environments and areas with high crime or vandalism.
  - Can record audio and video and store data automatically with location data allowing it to be easily later mapped.
  - Old devices work great, "look for [cameras] with 5 megapixels or more" (RFN, 2016, p.45).
  - Short battery life can be offset with an available connection to a network, solar panels and auxiliary battery packs.
- ***Satellite Imagery***
  - Excellent for capturing a type of rainforest data specific to a satellite system design.
  - Can be time-intensive to analyse large volumes of visual data.

    - ArcGIS, QGIS, Google Maps, Google Planet, other GIS and 3D globe softwares identified as key technologies for satellite imagery, mapping and visualisation.
- ***Maps***
    - "A powerful, visually engaging way to present [collected] information" (RFN, 2016, p.50).
    - Static, interactive, 3D globes and animated maps can be used to show more detail (zoom levels), changes over time and real-time location data.
    - "It is often easiest to use existing online interactive maps, which already include information layers such as roads and satellite images" (RFN, 2016, p.50).
    - Javascript libraries, Leaflet.js.
- ***Project Methodology and Software Engineering***
    - Don't assume, ask user groups (RNF, 2016) for what functionality is useful and for its reception. Ask and continue to review and adapt.
    - "Decide on critical features … before choosing [technology]" (RFN, 2016, p.18).
    - The use of technologies that can scale and are adaptable and flexible.
    - Have project time frames and spans with reasonable time for delays, problem-solving, and start early !
    - The development of a small scale pilot/ prototype to identify early strengths and weaknesses.
- ***Rainforest Environment Design***
    - Solar panels for remote systems.
    - Design for an under-reliance on remote communication networks and have a back-up plan. GSM, Wi-Fi, Bluetooth and local storage.

2.9 International Tropical Timber Council (ITTO) Summary Report

Meta-Evaluation of Previously Evaluated ITTO [Forest Information System] Projects

This Report, although from 2011, summarises the strengths and successes from past FIS projects and defines how to design and implement a future, more successful FIS that may avoid the pitfalls and mistakes of earlier systems and projects. The report highlights the need for a more refined, developed, systematic FIS that could collect, process and provide access to more reliable and more accurate information.

The aim of the paper (and requirements for an improved system) is for the long term sustainability of tropical timber forestry but also describes where a system could be improved to target illegal logging and how a system could be designed and applied for scientific research, investigation and the improved understanding of monetary investment and return in tropical forestry. The paper is from 2011 but sets a very interesting almost historical context for Rainforest Information System development.

The summary report identifies key areas for the development of a successful forest inventory system, including:

- Defining and identifying the value and type of the data that is input in to the system
- Clear aims for the system
- Closely understanding stakeholders
- Allocating enough time for technical and administrative obstacles.
- Limited and focused scope for the systems design
- Initially provide a base of minimum requirements, implement and then expand
- Modular systems that are extendible and maintainable.

An additional identified key, although not specifically relevant to this project, is to have a multidisciplinary, well trained and experienced development team with good working conditions, and in-depth understanding of the field of forestry and understanding of the technologies in use.

Project Keys:

- Outline basic modular technical subsystems for data collection, processing, database persistence, data presentation and for access to data.
- How to design a successful system.
- Limiting and focusing the scope of the system to an initial minimum base of requirements and functionality.

2.10 Open Forests

6 Principles that Help to Design a Robust Forest Information System

Open forests provide information and tools to improve forest computer systems and to reconnect people with forests. Their web article identifies some keys to making a successful system, although less formal than the previous sources, the article, almost blog, provides valuable qualitative insight and experience and is so included here in the literary review.

A summary of the article and its principles:

- Having a data structure that aligns with the business process or scientific purpose
- Iterations over waterfall
- User stories over feature lists
- "A modular approach [works well] for organic growth" (Watson, 2018).
- Confidence that system components should be "designed as simple as possible" (Watson, 2018).
- Avoiding vendor lock-in, open source can be more technical but more flexible.
- Web-based

Although not a peer reviewed research paper the articles consistency with key points from other sources is reaffirming.
The information is clear and written in my opinion from experience and the perspective of how to add significant value to a project and system.

2.11 WildLabs

The First Online Conservation Technology Community Network
Next Gen Wildlife Tracking: Extended Discussion Youtube Virtual Meetup

Wildlabs is an online community to share ideas, resources, code, projects and information between environmental conservationists, engineers, research scientists and hobbyists originally funded by Google and ARM to promote the discussion and sharing of conservation problem solving ideas.

Two main project keys from this source were the small but very interested multidisciplinary community in the field that, certainly in some ways, defines the field's current context.
And the second, from a useful member, was a very useful and thought out online resource software comparison table that includes the Forest Watch application mentioned earlier.

Project relevant highlights are summarised in my own comparison table of software applications with another tool from the Humanitarian Nomads comparison software solution finder, the Rainforest Connection Guardian application and another recommended by the Wildlabs community, Cyber Tracker.

It is also worth mentioning a Wildlabs youtube virtual meetup I participated in with members from the International bio-logging society, research biologists and a systems engineer discussing and interested in the future application of alternative satellite technologies such as CubeSats (lower-cost, smaller, easier to design and deploy satellites) and ICARUS (the International Cooperation for Animal Research Using Space) to monitor and retrieve information from natural environments gave me the idea to look into designing and launching my own satellite, but the cost, complexity, restricted radio band allocation regulation and the earlier identified 'project key' from the ITTO to limit the focus of the project made this idea out of scope and too expensive, but maybe not for future projects!

2.12 Humanitarian Nomad

Humanitarian Nomad Online Selection Tool

An excellent online selection solution tool to recommend from 50 different data collection/information management smartphone applications designed to collect field data with the aim to connect charitable aid organisations with the latest information management tools that are based on a user's inputs to different requirements and questions.
After submitting answers to questions similar to 'data format needs', 'selecting a prefered operating system' and 'is there a mapping spatial data requirement' a selection and overview of appropriate tools are displayed.
For this project's aims, there weren't any applications found that met the project's core requirements, (one of which is to develop an application) but the overview and comparison of the different applications was useful for familiarising with tools to input, process, network and organise form submitted field data. The Magpi application was included in the comparison from the 50 as it is the most widely used.

2.13 Software Application Comparison

Software Comparison of Field Data Collection, Networking and Monitoring Applications

A data collection software application comparison of free and easily accessible applications that are similar to the input TPS component of an Information System. The applications for comparison are taken from the previous sections in the literary review.

	Global Forest Watch (ForestWatcher)	**Rainforest Connection Guardian**	**Cyber Tracker (Wildlabs recommendation)**	**Magpi (Humanitarian Nomad selection tool)**
GSM	No	Yes	No	Yes
Wi-Fi	Yes	-	Yes	Yes
Offline	Yes	No	Yes	Yes
Type	Deforestation, Forestry, Events, Alerts	Illegal Logging, Poaching, Bioacoustic species	Environment monitoring, Biodiversity, Traditional tracking and	Form data, Alerts, Messaging.

		identification and monitoring	traditional animal tracking. Logistics, stock and warehousing levels.	
Functionality	Near real-time view of forest information. Navigable map. Upload new forest information. Works offline. Forestry Monitoring and management.	Automated streaming of audio forest and location information to the Internet via the GSM network.	Translates traditional tracking techniques to a digital format. Animal and tracking data field collection and networking for analysis and mapping. Data entry. Stores offline.	Data collection and upload. Electronic forms. Stores offline. Data Visualization. Form validation.
Data	Forms, Photos, Video GPS, Time, Custom Forestry Only	Audio stream, GPS	Animal Species, Custom, Ranger Patrols, Symbols and text	GPS, pictures, audio, text
Smartphone	Yes	Yes	Yes	Yes
Special	Interactive Map Near real-time with Internet connection Open source	Retrieves data only Open source	Designed for literate and illiterate users freeware	Built in mapping, Auto generated reporting, and filtering, Designed for literate and illiterate users
Cost	Free	Free	Free	Free
Users	Rangers, Foresters, Research Scientists, Biologists, Botanists.	Rangers, scientific monitoring and research.	Rangers, Animal trackers, Biologists	The most widely used mobile data collection application ~60,000 users, ~170 countries
Storage	Local, online cloud database	Cloud database	Local, networked computer	Cloud, REST, Web Service, API
Skill of use	Moderately technical	Automated	Easy to use once set to ecosystem and language	Comparatively easy data entry

Table 2: Comparison of field data collection software applications

2.14 Smart Devices

A Brief Overview Comparison of Smart Device Hardware

Fitness Activity Trackers, Smartphone and Tablet hardware features are over generalised in this table for a more simplistic overview comparison. The information was sourced from Google searches, different comparison websites, most notably, PhoneArena.com.

	Fitness Activity Tracker	**Smartphone**	**Tablet**
Price (recycled)	~$20 to ~$300	~$40 to ~$1500	~$50 ~$2500

Battery (new)	A few days to a year.	A few hours to several days.	A few hours to a few days.
Weight	~15g	~150g	~500g
Size	~1 inch	~5 inches	~8 inches
Processing power	Low	Medium	High
Internal Storage	Medium	High	High
Weather protection	High	Medium	Medium
Sensors	Accelerometer Altimeter Heart Rate Blood oxygen Compass GPS Ambient light Sleep Time	Accelerometer Gyroscope Compass GPS Biometric Temperature Humidity Ambient light Barometer Pedometer Time Camera Microphone	Accelerometer Gyroscope Compass GPS Ambient light Time Camera Microphone
Networking	Wifi Bluetooth USB	GSM Wifi Satellite Bluetooth USB	GSM Wifi Bluetooth USB
Operational Temperature	~0° to ~ 35°. With extremes of -20° to 45°	~0° to ~ 35°. With extremes of -20° to 45°	~0° to ~ 35°. With extremes of -20° to 45°
Operational Relative Humidity	~30% to ~60%, Waterproof	~30% to ~60% With extremes 5% to 95%	~30 to ~60 With extremes 5% to 95%

Table 3: Comparison of smart devices

From the comparison table notable characteristics could be identified:

1. Fitness trackers share a similarity with the animal tracking devices evaluated early in the literature review and typically express the characteristics of smaller streamlined smartphones.

2. Tablets share a similar crossover with smartphones but with more powerful processing.

The price, weight and size naturally increase with tablets having typically less battery life.

3. Smartphones, in general, have the most networking and built in sensor hardware options, with all devices having 3 main types of sensors; motion, environment and position that are capable of "providing raw data with high precision and accuracy" (Google, 2021).

It was concluded that all hardware devices, in general, appear to be capable of retrieving information from tropical rainforests when operating within their extreme temperature and humidity ranges (tropical rainforest conditions for the Amazon average ~28°c and 88% humidity (WWF, 2020)), and with additional confirmation from early literary review sources, particularly from the RFCX internship, that modified smartphone devices were shown to be the ideal hardware for the Input Component of the prototype Rainforest Information System and that a tablet computer could be used as a logical and viable choice for the Central Processing and Data Storage Component field computer's hardware.

2.15 Literary Review Conclusion

The Literary review provides an investigation of the current context of organisations, approaches, RIS's, FIS's, environmental technologies, hardware, project methodologies, system architecture, open source applications and smart devices that can be used to retrieve, network and present field data and information from a range of wildlife environments, ecosystems and tropical rainforests that give a clearer insight for formalising key requirements for both the functional and non-functional requirements of useful, effective and desirable features within a low cost and accessible prototype RIS.

The viability of using free open source software and recycled hardware was researched, particularly from the RFCX internship, and a clearer understanding of the field, including the aims of organisations, innovative approaches and the success of applied technologies, comparison and review of existing systems, and their advantages and disadvantages provided the foundation of information and insight for the design, implementation, choice of project methodology and baseline metric for the testing and the evaluation of performance of the prototype system and project.

The software and hardware comparisons above gave a short and generalised overview for the cost and availability and viability of open source applications and recycled smart devices for use as the hardware in the prototype system. With free open source software and recycled hardware showing comprehensive functionality and viability with few apparent disadvantages.

The above literary review provides a justification for the objectives and deliverables of the project and in addition helped to adjust the scope of the project.

2.16 Methods Introduction

This section details the selected software engineering project methodology, implementation methods, project plan, requirements analysis, task analysis and the formalised functional and nonfunctional requirements for the prototype RIS.

2.17 Project Methodology

Keys for identifying a suitable project methodology that could be applied in the project's analysis, design, implementation, testing and evaluation phases happened in few stages, and with a few additional considerations:

1 Choosing or adapting an existing methodology taking into consideration my own experience and expectations.

2 Choosing or adapting a methodology that could meet the aims and objectives of the project with a focus on decreasing the cost and increasing the accessibility, usability, desirability and value of the prototype so that users of existing systems might find the prototype, or its features, more useful, usable or an improvement over existing systems researched in the literary review.

3 Incorporating a requirements engineering process drawn from a task analysis and from the literary review identified 'project keys' that could define both the functional and non-functional requirements for the prototype system.

4 Choosing a methodology, or implementation methods, that would produce artefacts in a way that their features could be re-shaped or be re-developed to more closely align with the requirements and user needs of the system based on feedback and evaluation that might identify and be able to improve early strengths and weaknesses.

As a result, the below overview of the iterative development cycle (taken from the Interaction Design module of the degree program) was decided as the basis for the project methodology and for the implementation approach for the technical component artefacts.

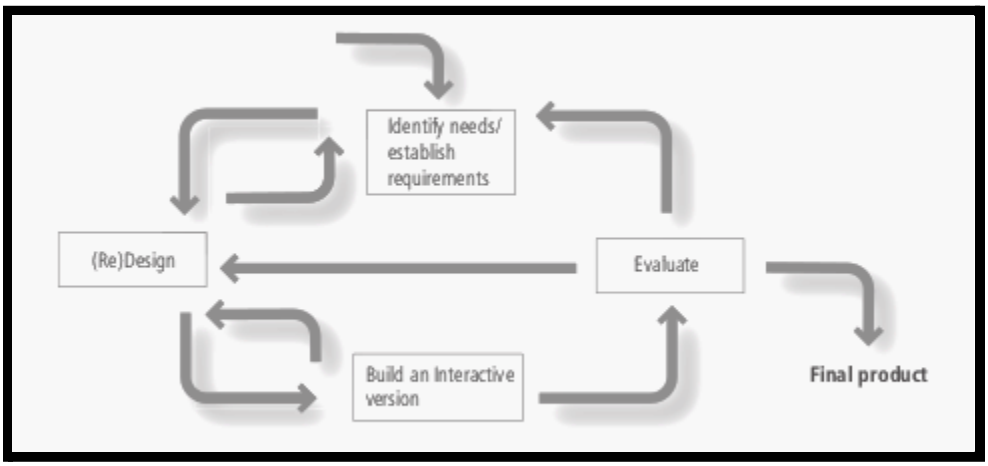


Fig. 8: Interaction design iterative development cycle (Murray, 2010, p.38).

In detail:

Identifying Needs and Establishing Requirements

It was decided that with a 'task analysis' to identify, actions, needs and functionality of end users and from examining and evaluating existing systems based on the research in the literary review together with use case models insightful requirements for a low cost, more accessible, possibly innovative prototype system could be formalised and requirements could be made more clear and be better documented.

(Re)Design

The design and redesign of the system architecture and individual components in subsequent iterations could be derived from new information, testing, feedback and evaluation.

The System Architecture and individual component design could be based on defined use cases and scenarios captured from the task analysis and the literary review identified 'project keys' with supporting UML diagrams to detail core use case tasks and system complexity graphically and accurately.

A database schema and formal development approach (detailed in the database design and implementation section) for the database and data model could be taken to ensure that the data model and contents of the database would align to the scientific purpose, and provide the necessary functionality of an RIS, as well as being more accurate in its implementation and operation. PostgreSQL was chosen for its suitability with scientific data.

Build an Interactive Version

Taken from Nielsen's 'Guerrilla HCI' (1994) taking an approach to make rapid prototyping faster and more effective, by developing a prototype system or unit that only needs to focus on a subset of a system's full functionality. The breadth or depth of a system's features could communicate an idea in a shorter time frame that can be reviewed more times by users within the project's overall time frame.

Implementation of a breadth or depth of an individual prototype component or functionality could be evaluated and redesigned more times and more closely developed to build on identified strengths by incorporating more immediate feedback.

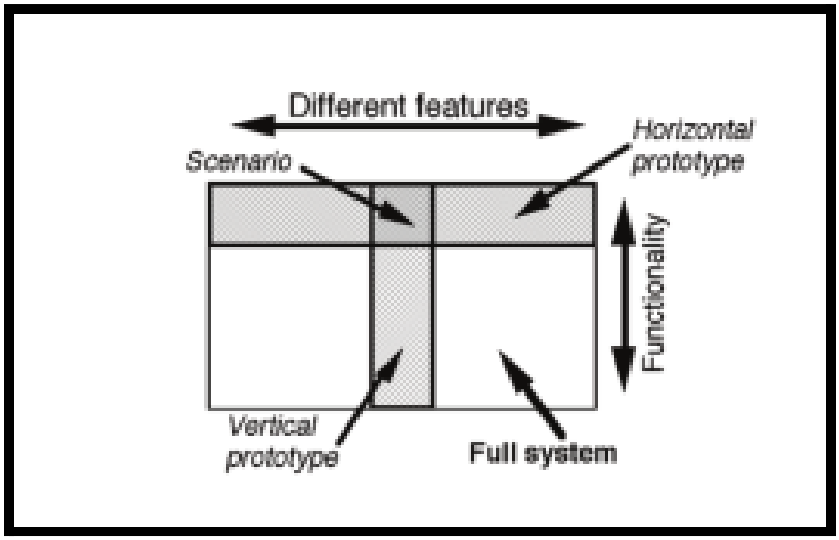


Fig. 9: Making rapid prototyping simpler with vertical and horizontal prototypes (Nielsen, 1994).

Evaluate

The testing, evaluation and validation of individual component artefacts and together as a complete system, including, unit testing, integration testing, whitebox testing, lab testing and field testing for both the technical and nontechnical aspects of the system, including accuracy of performance and usability that could either lead to the starting point for a next iteration or the final individual artefact or the completed prototype RIS.

The process model of the chosen project methodology also admittedly gives the possibility to revisit, correct mistakes and refine previous milestones; a very useful quality for my first attempt at an academic research project.

2.18 Additional Project Methodology Concepts and Implementation Methods

In addition to the above project methodology some other concepts were applied and found to be very useful in completing the project. Some sections from the 'methods section' in the PPR are included in this section as they very accurately describe the methods that were implemented in the project.

Using an Information System Structure

A key decision in this project was to use the familiar Information System structure from the degree program, of an Input (TPS type) Component, a Processing and Database Storage Component (web server and database) and an Output Presentation Component (a web page based dashboard) for the system architecture. Helping to divide the project into smaller, modular and easier to work with and design, implement, test and evaluate artefacts and components that together could be networked to make the complete Rainforest Information System.

Using the Information System structure improved the organisation and made possible the centralization of data including the management of its flow between the system's components. The IS components could also be made to parallel with the project's objectives and deliverables within the chosen methodology.

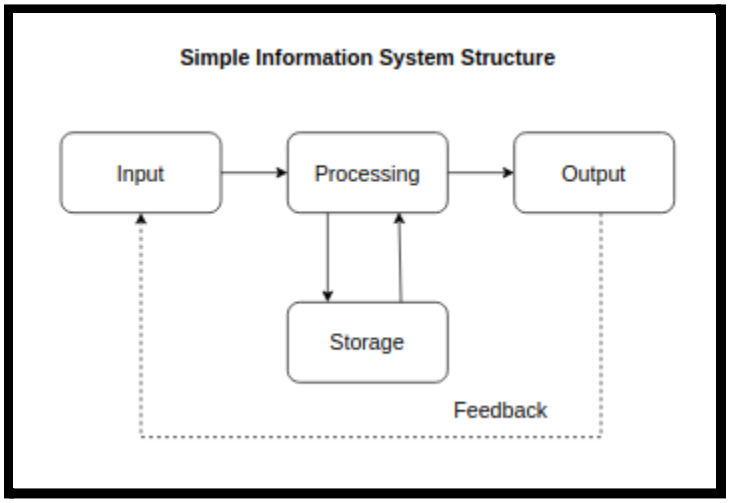


Fig. 10: Simple Information System Structure

With the above project methodology, the prototype could focus on the complete functionality (depth) of a limited set of features (breadth) of a complete system that together with the complete Information System structure could implement an IS feedback cycle, useful for testing, evaluation and iterative re-design.

With a limited breadth of coverage the idea of the system could be shown to the system's potential users, and with their validation, more depth of functionality, an alignment to a desired quality and additional development of features could take place.

Some important additional helpful notes that were made when studying the different modules of the degree program are also included as they were a great influence on the project's structure (I'm not sure of their exact source location within the different modules) :

- Use programming Interfaces → 'Any is better than none'
- Make early development architecture decisions
- Modular → easy control of complexity
- Easy to understand organisation and code
- Highly cohesive classes and methods → 'single little tasks'
- Low Coupling → minimal connection between modules
- Focus on key areas
- Meeting dates > requirements
- Stabilise requirements
- 'Diagrams too complex to hand draw are too complex'
- The 'fundamental problem of software engineering is the intangible complexity of code'
- A few simple Class Diagrams as a main deliverable.
- Have mostly <0, m> constraints in a behavioural diagram
- Objects and their methods should have rich behaviour
- The main interaction diagram to use is the Sequence diagram
- Distributed control is better than a central object
- Localise
- Better polymorphisms
- Diagrams can be un-useful only use a mix that are
- UML will be used in a refined sketch mode to give understanding and useful detail but not to blueprint the project.
- A good OO style uses a lot of little objects with a lot of little methods that give us a lot of plug points for overriding and variation.

Following a Hybrid of an Iterative Agile and Waterfall Software Development Life-Cycle Model <Extract from the PPR Section: Methods>

Ideally a completely Agile iterative development approach would be used but realistically the project may progress more successfully with a hybrid between a Waterfall and Iterative model.
By dividing the development into the Waterfall phases : Analysis, Design, Implementation and Testing, that are progressed sequentially, together with certain components being conducted with an Agile iterative approach, will mean areas of the project can be completed definitively within the waterfall stages as functioning components of the project and system that are ready to meet the aims of the project, and to give other components time, structure and retrospectives to be able to make additions and improvements in later iterations.
The iterative approach applied to the Android Application, for example, will prioritise important functions of the system and their completion first, resulting in definitive objective-meeting functions that later iterations can expand and improve or add additional functionality to.

Testing <Extract from the PPR Section: Methods>

Testing the software, hardware and the system technically, and as a complete Rainforest Information System will include :

- Usability testing - for its ease of use and operation.
- Unit testing - to check modules for bugs and internal states and working.
- Integration testing - that the parts of the system work together.
- System testing - that the system meets the requirements specification.
- Acceptance testing - validation from the end user.

Hardware was to be tested in tropical environment conditions to see how they perform.
Black box testing would be ideal and will be sought from colleges but as the project is self-study most of the testing will be white-box where the technical details, modules, components and code will be known by me but this will improve my technical understanding (that is technically an objective of the project).

2.19 Project Plan

The project plan helped to define the scope, identify key phases and tasks and organise the time available, and keep the project on track.

> "large projects overrun, on average, by 50% of the original estimate"
> (Blackwell, 2007, p.55).

The project's plan was allocated to have additional reserve time for this very reason and due to the expectation that many of the technologies and concepts used were new and the expectation that bringing many new components together and applying different areas from different subjects into one academic research project and complete Rainforest Information System could take more time than expected.

A schedule detailing the tasks completed, and needed to be completed, for the project to meet its aims and dates is shown below in the Gantt Project Management chart that was informally planned and written at the beginning of the project but formalised into the Gantt chart for the Preliminary Project Report submission date.

The Gantt chart is included at the end of this section and an online version can be found at:
https://github.com/SatellitesAndRainforests/Rainforest-Information-System-Documentation

The project plan is quite accurate in retrospect of the actual project's workflow and its completion dates. The individual tasks are annotated well in the chart and so not explained in as much detail in this section. Tasks were kept necessarily open-ended in certain areas where estimates for time frames or additional detail would not have been useful, but were included to be a helpful illustration of the project's management plan.

Key milestones, project stages, component artefacts and individual sub-tasks are indicated in different colours. Larger tasks were broken down into smaller tasks with the advice from the project guide to not allocate more than a maximum of 2 weeks duration for each task. Maximum time was made available by starting the project as early as possible following the exams of the previous year. Ongoing documentation was made throughout the project and proved very useful when writing up the reports.

Admittedly some phase sub-tasks were revisited more times than detailed in the chart following the iterative overview methodology detailed in the previous section and did not follow the exact more linear sequence detailed in the chart, including, the formalising of the aims and objectives task, and many of the

component's development sub-tasks. Due to my interest in the subject, additional project research was completed outside of the timeframes and of what is detailed in the chart. There were also some necessary interruptions due to the quarantine and other work. The second iteration of the Android application was the only component that had 2 complete formal iterations. Time permitting, more iterations could be completed.

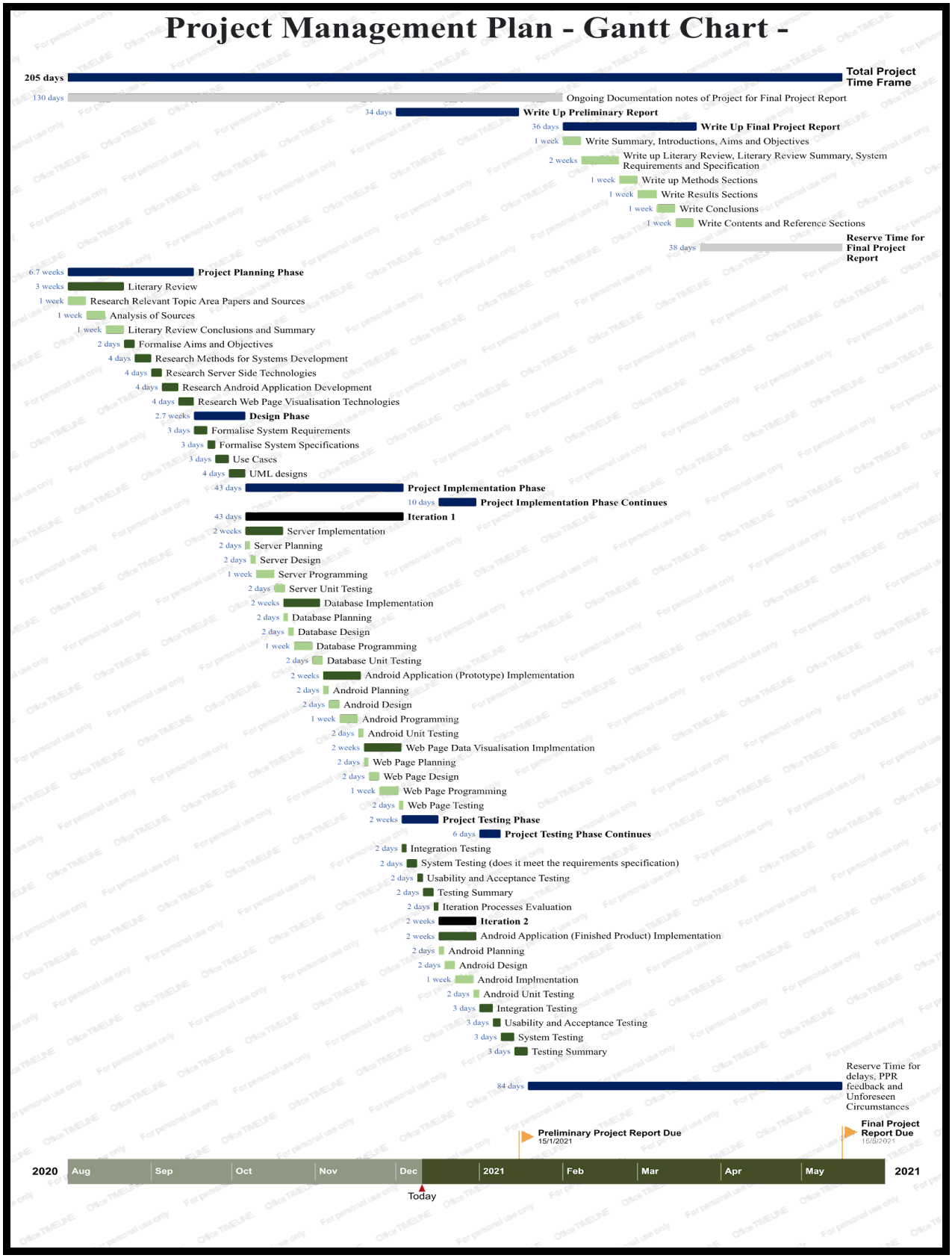


Fig. 11: Project management plan Gantt Chart

2.20 Additional Research and Interviews

A Lot of data is needed for quantitative analysis and for statistical significance, and due to this project being conducted in self-study and during a global pandemic ! It was a necessity to explore other avenues to gather relevant data and feedback that could be used to help formalise the requirements and align the development of the prototype to meet the aims of the project.

As an alternative, qualitative data and insight could be gathered from a much smaller sample size and still be advantageous in comparison to a larger sample size in that the insight it would provide could be more insightful in different ways.

As a compromise it was possible to conduct semi-structured interviews with a less formal Q/A structure within job interviews I attended for software engineering positions (online) with :

- Cambridge Environmental Research Consultancy
- The Natural History Museum (for a software engineering position)
- The United Nations Environment Programme World Conservation Monitoring Centre (UNEP-WCMC) (also in Cambridge)
- A.D. Little (a City of London international consultancy firm)
- Rainforest Connection (Interview, Internship and in additional online meetings, including with Topher White the founder and CEO)
- TreeApp (City of London based marketing company that promotes the planting of trees in tropical forest locations)

I was invited to interview with these companies based on my CV and the youtube walkthroughs prepared for the project (links included at the top of the report) that were completed before the time of these interviews and I was able to ask for feedback on the Input Component, Processing and Data Storage Component, Output Presentation Component and the complete system from experienced professionals within the field of software engineering environmental scientific applications and systems.

The feedback received helped the project by initiating a more careful task analysis for an RIS and implementing some additional improvements to the prototype from the feedback received, including a focus on task context (where the tasks of rainforest data collection are performed and how the tropical rainforest environment may affect the task and technology), using a Docker based database, cleaning up some Java code notably in the Android application and the JDBC interface between the server and the database and improving the, literary review, testing, and the evaluation phases for this report.

I do however have some of my own personal tropical rainforest experience from travelling to New Zealand, Australia, Japan, Mexico, Belize, Guatemala, Honduras, Costa Rica, Nicaragua, Fiji, Colombia and Peru but additional research and user testing for this project in these areas I felt was compromised.

As a result the requirements specification was primarily based from the research of the literary review with the addition of the revised task analysis and the greater consideration for the context of an RIS from the internship, interviews and meetings I attended.

It was an important part of the project to better understand the stakeholders and users of scientific environmental systems, including system architects, technical CEO's, developers, founders and investors, project and hiring managers.

As the project progressed the requirements, objectives and other sections were slightly adapted, updated and revised based on the feedback received from these interviews.

2.21 Rainforest Information System Task Analysis and Scenarios

Traditional field biologist approaches to working in the field aim to measure an aspect of the rainforest such as measuring biodiversity, capturing and tagging animals with simple or sophisticated collars (as explored above) and taking data entry surveys within an area or location, such as a tree, a quadrant or net at a given time for a given duration.

Different types of technology can improve the effectiveness of these field data collection tasks. Canopy fogging for example sprays a low dose of insecticide to an above tree in the rainforest canopy and collects animals that fall into a large screen that can then be more easily surveyed.

The automation of the traditional task of field data collection can be improved with applications of electronic sensors that can retrieve comprehensive sets of data with different types, grades and calibrations of sensors and with different levels of sampling, precisions and accuracy.

Innovations in electronic sensors and their applications can increase the effectiveness of the data-entry task, for example, a sophisticated sensor could take a photo or thermal image of the tree, if the application could accurately identify animals and insects without the need for fogging and traditional techniques.

Automation of these processes can significantly increase the speed and reduce the complexity of data collection tasks.

With the use of scenarios and use-cases for users of the ‘data capture’ and ‘analysis’ tasks of a Rainforest Information System the identification of tasks can then map to requirements, designs and functions within the prototype system.

Scenarios

Scenarios for the generalised ‘collection’ and ‘analysis’ tasks of field data from a tropical rainforest using sensors in an Input data entry component and the evaluation of the data with an Output presentation component of a Rainforest Information System:

Automated Field Data Collection with a Inbuilt Sensor application
A user can initiate an application to collect field data with its inbuilt sensors.
An acknowledgment confirmation message is communicated to the user.
Automated field data entry continues.
The user can stop the field data collection, the data saved locally and an acknowledgment communicated to the user.
The user can instruct the application to send the data to another computer, not in the field.
The user can delete the local data.

Viewing and Analysis of Field Data
A user can view a presentation of the retrieved data.
The user can interact with the retrieved data.
The user can incorporate the retrieved data with other retrieved data from the system and other systems.
The user can map the data.
The user can see all the sensor data mapped and interact/ select different compositions of the field data.

The user can interact with the map to view the field data from different perspectives.
The user can add to or clear the selected data from the screen.

Further illustrated in a simple use-case diagram of a complete system :

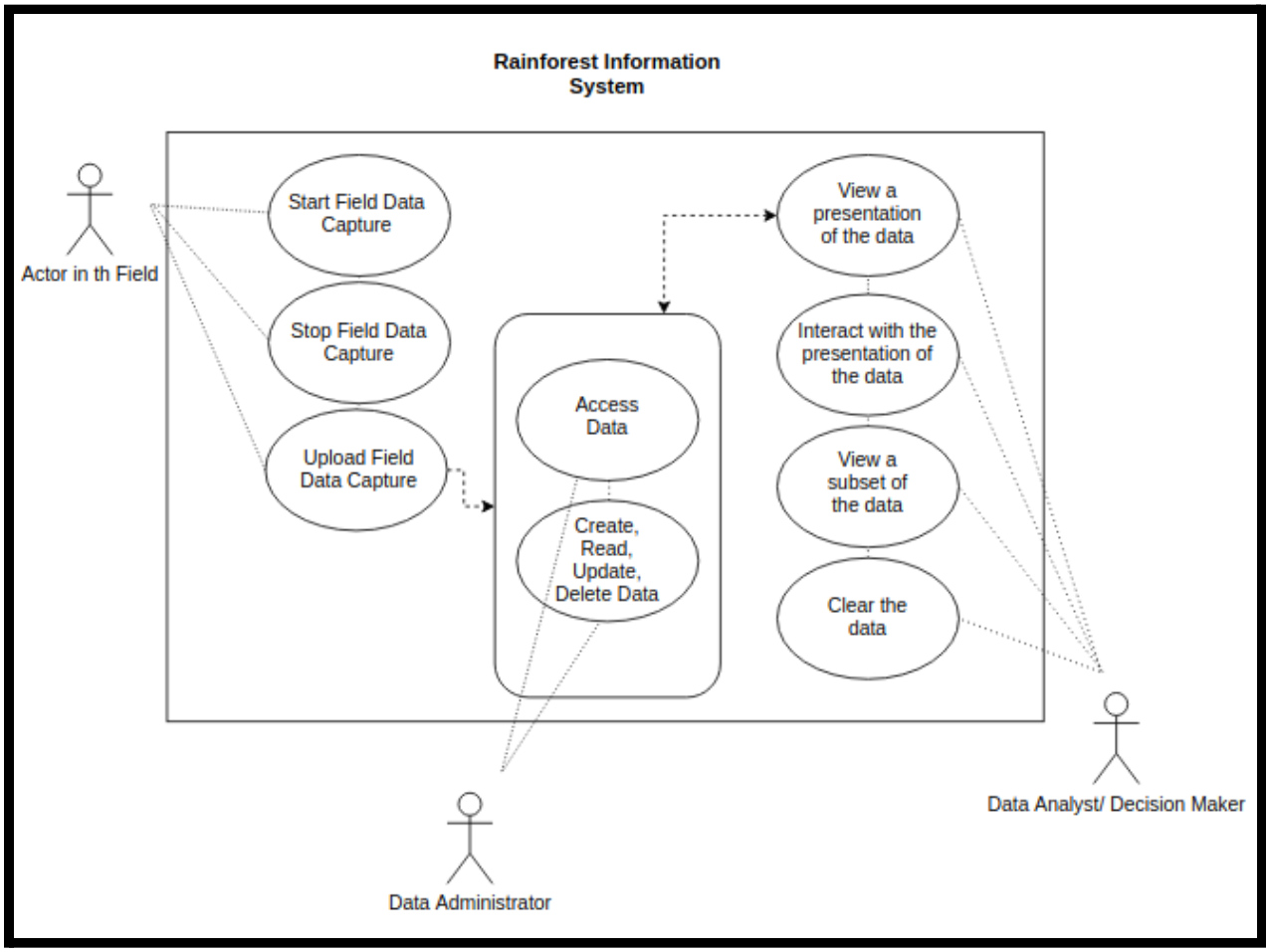


Fig. 12: Simple use-case diagram of the complete system

2.22 Requirements Specification

From the literary review, 'project keys', internship, interviews and the following task analysis it was possible to define many of the software, hardware, usability, functional and nonfunctional primary and secondary requirements for the prototype. Requirements are prioritised between necessary (primary) and secondary to organise the value of initial iterations.

Project Specific Requirements

- Develop a small scale pilot/ prototype quickly, of limited scope, of a base of minimum requirements with some vertical and some horizontal functionality to exhibit ideas and identify early strengths and weaknesses and iterate, expand and refine based upon feedback, testing, review and the evaluation of the performance of the system.

System Specific Requirements

- The Automated data collection of a variety of accurate input field data from a base set of sensors from a wide range of locations within tropical rainforests.
- The networking capability of field data from an input component to a computer also in or not in the field.
- A web accessible and interactive global view of the retrieved data for data analysis and evaluation of the system's accuracy and performance.
- A battery-performance balance that considers sample rate, accuracy and the volume of data retrieved.
- Data that is easily accessible for evaluation, review and feedback.
- Use of recycled hardware and open source free software to reduce costs, increase accessibility and to make use of the ubiquity, mobility and availability of smart devices

System Architecture Requirements

- Complete Information System structure
- System components designed as simple as possible
- Persistent centralised database storage
- Flexible, maintainable and modular
- Solid Design

Hardware Requirements

Primary

- Smart device with inbuilt networking channels, GSM, Wifi, Bluetooth, SMS
- Fallback transfer of data capability; USB/ memory card removal.
- Easily accessible and available that can scale
- Recycled, low cost, low target for theft and vandalism
- Base sensors : {GPS, Temperature, Humidity, Time}
- Local internal data storage
- Portable and lightweight
- Works offline
- Reasonable processing power
- Operational temperature range : -20°c to 45°c (rainforest average ~28°c)
- Operational humidity range : 5% to 95%

Secondary

- Additional implementable sensors: {camera, microphone}
- Remote location solar panels

- Auxiliary battery pack
- Additional weather protection and insect proof casing to be able to operate in tropical rainforest environment conditions.

Software requirements

Primary

- Automated data collection function
- Make use of satellite imagery
- Open Source, free of vendor lock-in, community accessible
- Linux based operating system (free and open source)
- Can scale
- Web based
- Internet and GSM network access and data upload functionality
- Offline data storage
- Works offline
- Manual data entry functionality
- Base sensor set implementation
- 3D globe or interactive map to present, visualise GPS coordinates and other retrieved field data and information from other sources (including time, date, and sensor readings)
- 3D globe or interactive map zoom levels, different map overlays (satellite, road, terrain).
- Functionality to select, and reset data retrieved

Secondary

- Built in data validation function
- Built in data filtering function
- Anomalous values filtering function
- Auto generated reports
- Additional sensor implementation, camera, microphone
- Illiterate and literate user design
- Functionality to customise data retrieved

Data Model requirements

- A data structure that aligns with the scientific purpose
- Base sensor set implementation
- Position and environment data types
- Extendible for additional data types, text, image, video, audio.

Additional non-functional requirements :

- Collects a manageable level of data
- Be able to operate in tropical environment conditions
- Be cheap
- Accessible
- Easy to use
- Data visualisation should be attractive and interactive
- As much of the system should be automated as possible
- Accuracy
- Error messages and confirmation messages should be informative.

From the literary review, interviews, internship and task analysis, gaps about the physical considerations, environmental characteristics, useful features and functionality for the intended system could be identified

that could increase the value, usability, desirability and accessibility of the prototype RIS with hardware requirements that would operate well in tropical rainforest conditions and that importantly would be more easily available and lower targets for theft and vandalism in developing countries of the world where most tropical rainforests are found.

2.23 Test Cases

A test plan and criteria for acceptance testing in a later phase of the project can also be derived from the above system and software requirements.

The requirements, scenarios and use-cases defined in this phase can be used to validate the prototype and evaluate that it can perform the use-cases of the identified user needs and for use as metrics to evaluate the performance of the resulting project end-state of the prototype.

Functional requirements can be mapped to use-cases and additional non-functional requirements allowing testing to be based on the use-cases and scenarios.

Use-cases and requirements can also be used in additional activities to validate more detailed designs that in turn can be used to verify components have been built correctly and in turn the detailed designs used in more formal testing plans for the evaluation of the resulting components and the complete system.

Ideally a review to validate the formalised requirements, scenarios and use-cases at this stage with close and increased involvement with different project stakeholders and end-users would help to maintain an alignment of the project's process which could continue throughout different milestones of the project's different phases including the planned incremental cycles of design, implementation and evaluation helping to evolve a more details design over time and could be used as a sort of quality gate to direct and progress future iterations reducing the risk of the project not meeting its aims.

Chapter 3
The Design, Development and Implementation of the Prototype

3.0 Introduction

The design, development and implementation of the prototype follows from the understanding of the current context of the field of tropical rainforest data collection, current technologies used in the field, their users, tasks and the formal requirements, use cases, aims and the objectives of the project.

The implementation methods used for each technical component of the prototype Information System each produce a technical artefact that when networked together make the complete Rainforest Information System.

3.1 System Architecture

The Information System architecture of the project follows a client-server model so that data can be centralised and many clients can access and contribute to it in a many to one relationship. This can be particularly useful with many input field data collecting components operating in many different rainforests around the world located between the Tropics of Cancer and Capricorn.

The centralised data can in turn be made available to many interactive web-based output clients for visualisation and analysis of the collected data.

3.2 Overview of the Proposed System - Rough sketch

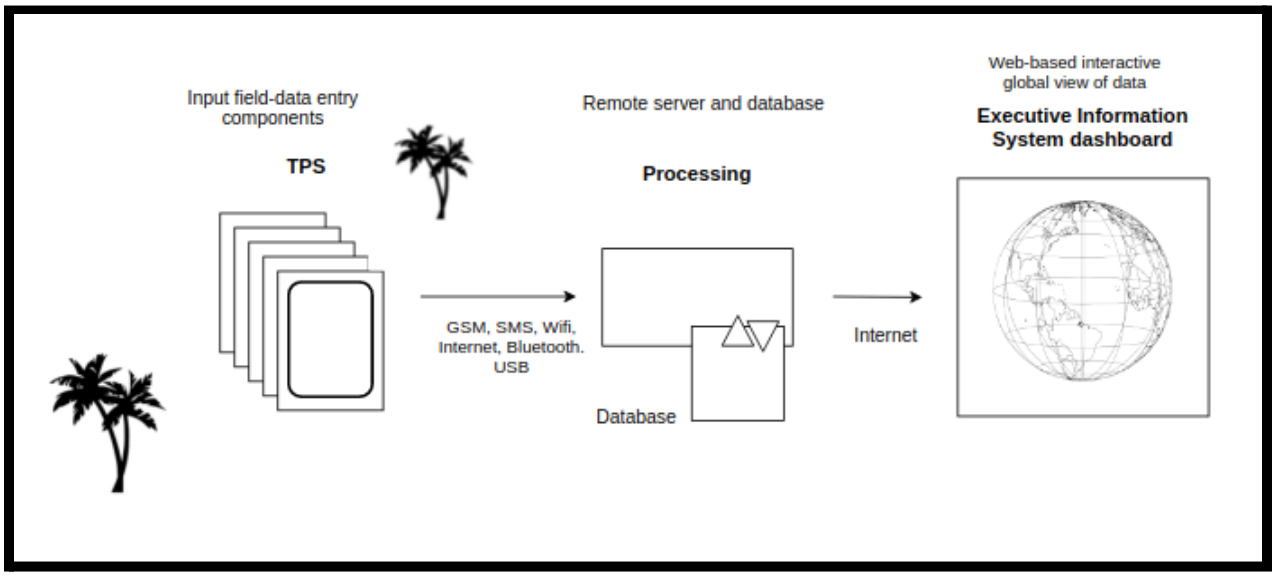


Fig. 13: System rough sketch

3.3 Track and Point Data Model

The data-model is identified to be a key component of the system. The data model is used to structure, store, process, visualise and network the collected sensor, time and GPS data through the different components of the RIS.

The data model chosen to retrieve, store and network the input field-data is a 'Track and Point' data model pairing sensor data with GPS coordinates over time based on the stack overflow posts:

"The best way to store gps coordinates (track) on Server" and,
"Floating point - Rails 3 Float or Decimal for GPS Coordinates"

The Track and Point data model is illustrated in the following section but described in greater detail in this section to outline its importance :

Many sensor data values for a single location and time can be stored in a single 'Point' with an accompanying unique ID, date-and-time and Latitude and Longitude values for where and when the sensor data values were recorded.

Each 'Point' data object can then be stored within a single 'Track' object with its own unique ID value, name and description for the particular session of gathering field data as identified in the task analysis and defined in the use case to capture field data input over time at the same or different locations.

3.4 GPS Precision

The circumference of the Earth is ~40,000km. Where a Longitude meridian (vertical) around the Earth intersects with a Latitudinal parallel (horizontal) is called a coordinate. A degree of a 360° parallel at the Equator equates to about ~110km but towards a pole of the Earth where a single Latitude parallel around covers less distance means that for a parallel at the equator (and a single degree) equates to a different distance (Rawkeen, 2012) than towards the pole.

Digit of Precision	Approximate area at the Equator	Approx area at the 48 parallel (Paris)
0	± 110 km	± 74 km
1	± 11 km	± 7.4 km
2	± 1.1 km	± 0.74 km
3	± 110 meters	± 74 meters
4	± 11 meters	± 7.4 meters
5	± 1.1 meters	± 0.74 meters
6	± 1.1 cm	± 0.74 cm

Table 4: GPS decimal precision

It follows that GPS coordinates within the Tropics of Cancer and Capricorn will have varying degrees of accuracy depending on how many digits of precision they are recorded to.
The table (left) illustrates the digits needed for reasonable accuracy. The equation used to calculate the approximate areas for the 48th parallel (Paris) and for the Equator is :

cos(48) * 110 (km, the approx distance of 1 degree) = 75 km

For the precision of the GPS coordinates needed for this system it was decided that 6 decimal places of precision would be more than adequate for the prototype.

3.5 Input Component

3.5.0 Input Component Design

Considered in some ways the most important component of an Information System as the source of the majority of input data used by the rest of the system, the Input Component's design was carefully detailed with:

- A more refined use case for the 'record input data' use case
- A noun phrase stage for the object-oriented class identification
- A UML class diagram to incorporate the Track and Point data model, define methods, data types, their private and public access, interfaces and the relationships between objects
- An Activity UML diagram to clarify the method interactions and functionality of the application for the use case
- And finally a simple prototype user interface (UI) for the application derived from the above.

3.5.1 Refined 'Input-Data Use Case'

Following the identification of the field data entry user task and the formalisation of requirements, a more refined use case could be defined and renamed for the design of the key functionally of the input component.

Record Track Point Sensor Data Use Case
Goal Level : Level 1
Main Success Scenario :
1. User (Agent) initiates tracking
2. User stops tracking (data is automatically saved locally)
3. User sends data to server
4. User can delete local data

Extensions :
3a. User is notified that a connection could not be established. The data could not be sent.

3.5.2 Noun Phrases and Class Identification

By identifying relevant nouns from a description of the prototype's requirements and functionality, key object-oriented classes for the programming of the input components applications could be identified. Relevant nouns are underlined:

> "The rainforest has an environment. The environment is complex and hard to explore. Data can be gathered from the environment. Different locations provide different types of data at different times. The locations can be connected to show a relationship between locations in a forest. Different sensors can collect different types of data. The technology gathering the data can be static, automatic, attached to an animal, or partially-automatic and can be operated by a user with interaction feedback communicated with an information message display. The data can be saved locally to a file. The gathered data

> can be networked to another location, a database. A system must collect, network, store persistently and make data interactive, presentable and available from a input component to a centralised server and database to a web-based interface 3D globe or map for the presentation of and interaction with the data"

With a following list of relevant nouns identified for the input component's object-oriented programming classes:

- Rainforest
- Environment
- Data — Renamed to (Point) following the data model design
- Location — Renamed to (Track) following the data model design
- Time — Typically a pre existing programming language object
- Sensor
- File — Typically an pre existing programming language object
- Networked — Renamed to (Client) following the system architecture design
- User
- Information message display

3.5.3 Class Diagram

With the identified classes, a UML Class Diagram incorporating the data model design could be drawn to formalise and clarify the design for the functionality of the new refined use case for the input component.

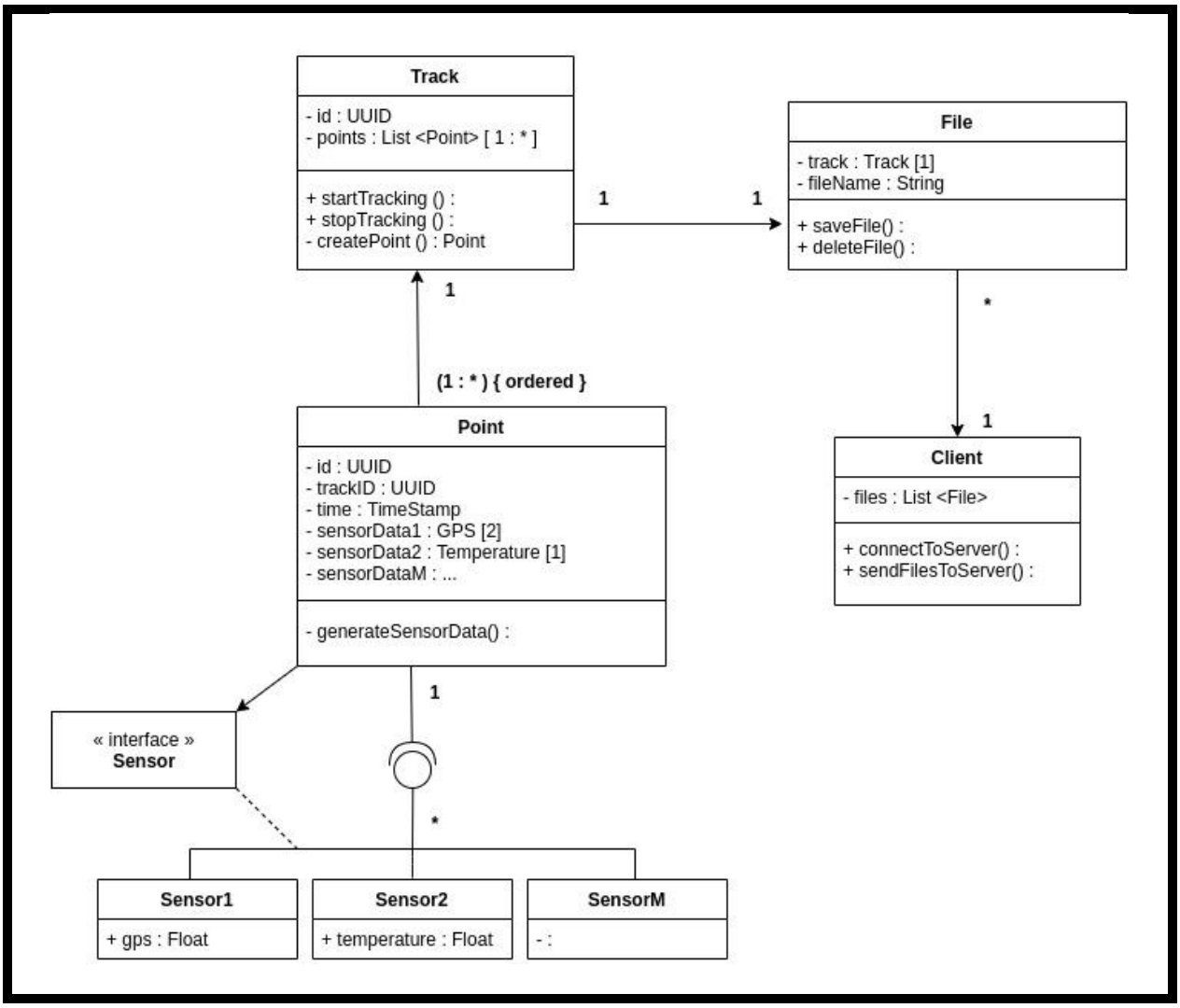


Fig. 14: Input component Class diagram

The Class diagram:

- Identifies methods from the use case and defines them in their relevant classes.
- Identifies appropriate data types for the field values of objects (e.g. UUID for unique ID values for each Track and Point object).
- Defines private (-) and public (+) access for each method and field variable based on expected interaction with other classes.
- Defines the expected relationship between classes (e.g (1:*) (one to many) {ordered}) for sequentially captured sensor data values to be captured at each interval and location for a moving device.

Some key design decisions made where to :

- Save one Track object with an internal ordered List of Point objects per File Object. This coincides with the initiation of the start data collecting tracking session, where new data Points are saved at intervals until the stop tracking event. This makes it possible to save, delete and upload individual sessions.

This was also considered a good design choice to make the data transfer, when uploading in less reliable networks, more robust.

- An interface was used between the Sensor class and Point class to make it easier in the future to implement more sensors into the Point objects. Although for clarity and simplicity each sensor is defined as a field variable in the Point class to detail the expected amount of values for the base requirement sensors (1 for temperature, 1 for humidity, 2 values for GPS Latitude and Longitude).

3.5.4 Activity Diagram

Following the Class diagram it was possible to define a Sequence Diagram to clarify the activity flow for the data capture use case with the identified classes, their relationship, structure and the Track and Point data Model.

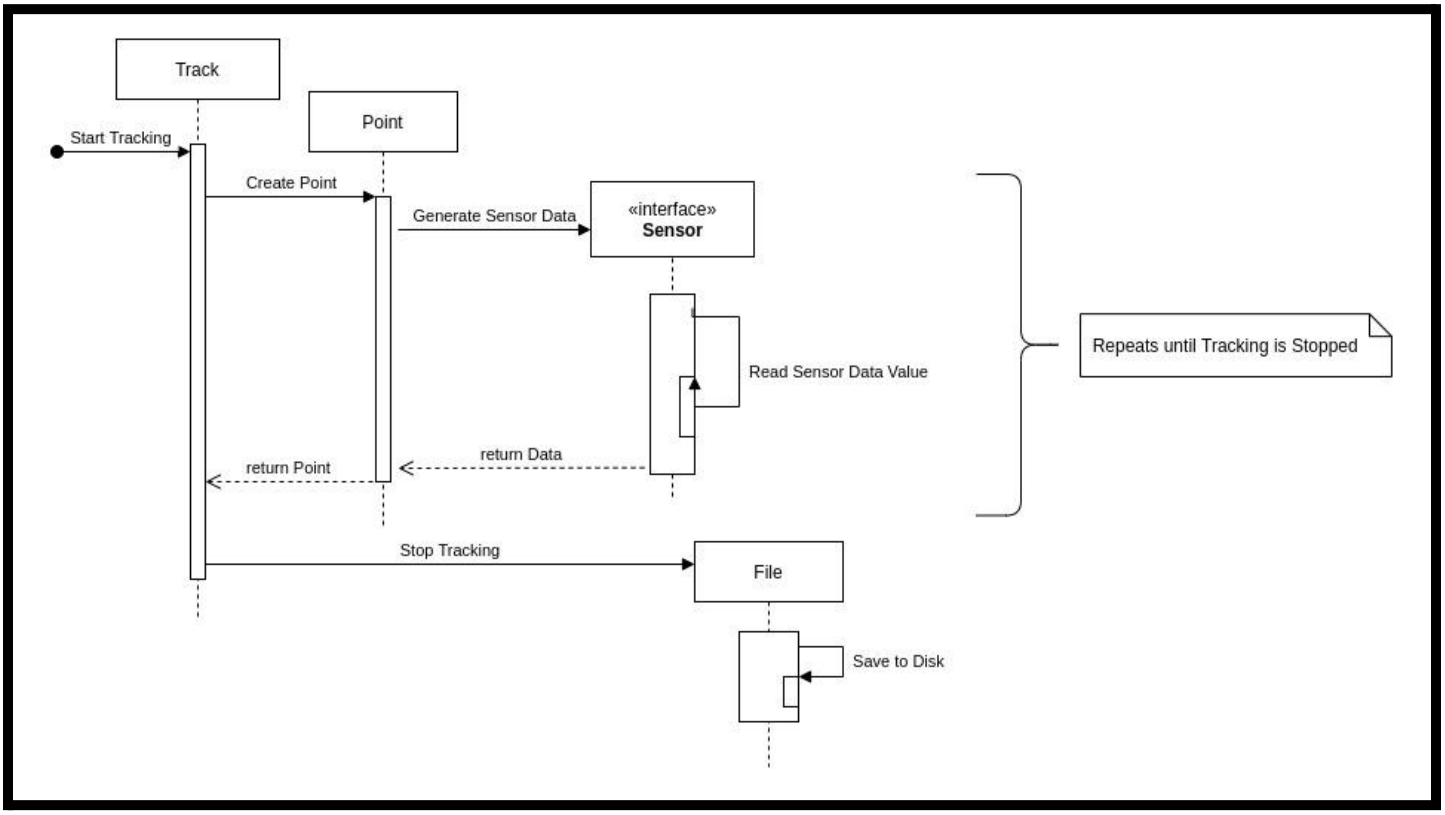


Fig. 15: Input component Activity diagram

The Start Tracking method would coincide with the initiation of a new Track object that initiates the creation of a new Point object, containing Sensor objects with data values retrieved from reading the hardware sensor's current data values.
New Point objects with Sensor objects are created at an interval (undefined at this design stage) until the Stop Tracking method is called from the user interface.
Each Point object is stored, as seen earlier, in the ordered List, ordered by their creation/ TimeStamp object within the Track object, that is then communicated to the File object to be saved locally.

Objects are then available to the Java garbage collector as they are no longer referenced within the program.

3.5.5 User Interface

A simple User Interface design could then be drawn from the use case, Class Diagram and the Activity Diagram :

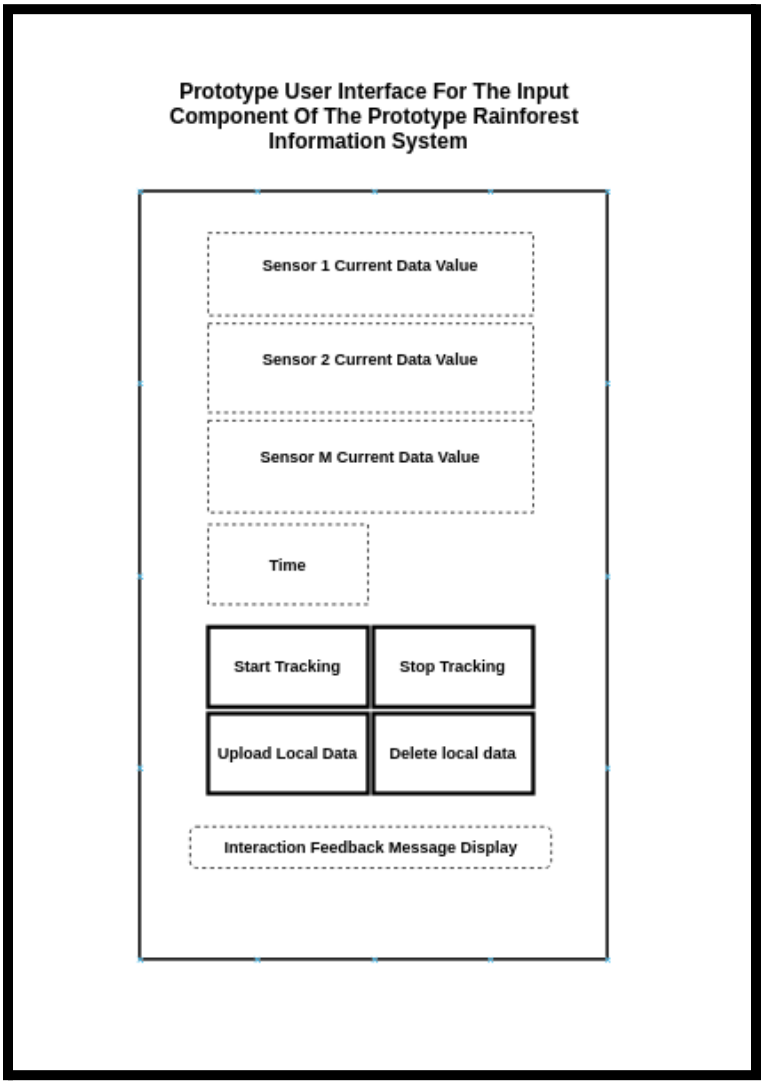


Fig. 16: Input component prototype user interface

Start Tracking, Strop Tracking, Upload Local Data, and, Delete Local Data methods/ functionality coincide with event triggering buttons (in bold outline) and current sensors data value readings, the time, and the feedback messages to display and communicate interaction feedback to the user within the dotted lines.

The interface was designed very simply to increase the speed of the component and prototype's development and to be able to present to users more quickly within the rapid prototyping methodology to elicit more immediate feedback to identify early strengths and weaknesses before more comprehensive choices could be made for later iterations.

3.5.6 Hardware

The smart device hardware choice was relatively straightforward as surprisingly very few smartphones have inbuilt humidity sensors (one of the required base sensors to implement). It was decided to use a smartphone over using an activity tracker, or other smart device, due to the greater processing power, storage and amount of inbuilt sensor and networking components as researched in the literary review over activity trackers and tablets as well as the perceived ease and support for development of an Android application that are mainly written in Java (the main programming language of the degree program).

A key aim of the project was to test the viability of using recycled smart devices so there was a key requirement that the smartphone should be several years old.

A few Google searches revealed that less than 30 smartphones were found to have the required inbuilt humidity sensor (phoneegg.com, 2021) and following an additional Google search for smart devices with the most inbuilt sensors, the 2013 flagship Samsung Galaxy S4 was found to be less than $50 dollars on eBay, had the most inbuilt sensors of almost all smartphones at the time (gsmarena.com, 2013) and considered one of the top 20 best selling mobile phones of all time (Telegraph, 2017), increasing the prototypes accessibility, and met all the other hardware primary and secondary requirements for the Input Component, system architecture and the project.

In relation to the primary and secondary requirements:

- The 2013 Samsung Galaxy S4 smartphone has the required networking channels (GSM, Wifi, Bluetooth and SMS).
- Has a removable micro SD storage card and a micro USB access port to be able to implement the required fallback and offline data transfer and can be powered by the same port in the field by an auxiliary battery pack.
- Is easily accessible being one of the top 20 selling mobile phones of all time (TheGuardian.com, 2017) and available from ebay for ~$50 reducing its profile for vandalism or theft.
- Includes the required base GPS, temperature, humidity and time sensors with additional implementable sensors including: infrared gesture, proximity, gyroscopic (3 axes), accelerometer, geomagnetic field, barometer atmospheric pressure, cover and RGB light intensity sensors.
- Includes a 13 megapixel camera (over 6 was identified in the literary review to be adequate) and a microphone.
- Has a Linux based operating system (Android).
- Is portable and lightweight.
- Has a more than adequate 1.4 Ghz processor with 2 GB of RAM.
- With an operational temperature range of the required -20° to 45°.
- And operational humidity ranges between 5% to 95%.

A less popular more rugged variant, the ‘Samsung Galaxy S4 Active’, was also identified with a greater level of waterproofing, dust and shock protection but it was decided for the prototype to evaluate the performance of the more popular version to test the use of a more general smartphone for use in tropical rainforest conditions and maintain a high level of accessibility for the input component’s hardware.

3.5.7 Input Component Implementation

The Smartphone Android Application and the Graphical User Interface

The applications source code is available in 'Appendix A' and online at the projects Github project page :

https://github.com/SatellitesAndRainforests/RainforestApp

As a key component of the Information System, that provides the majority of the input data for the rest of the system, the iterative rapid prototyping approach for the component's implementation would enable a faster presentation of the component to users to be evaluated, tested, and a iteration retrospective could be conducted and that the findings could be the basis of the analysis and design stages of the next iteration to improve this key components performance in the complete system.

Between the iterations, user testing, unit testing, system integration testing with the other components of the system and the feedback loop of the complete RIS provided source material for the iteration retrospective.

The initial iteration of the Android application focused on the prioritisation of the set of all primary requirements for the component as identified in the requirements specification and clarified in the design phase.

By first implementing the core requirements of the component it was easier to meet time allocations and to provide the functionality for the other components during their design, implementation, testing and evaluation phases.

The below diagram illustrates the evolution of the graphical user interface (GUI) through the two iterations. The first illustrates the initial rapid prototyping implementation of the core functionality of the input component's use case, and the second applies and resolves the findings of the unit, user, integration and system testing.

It may be useful to recap the input components overview video at this point.

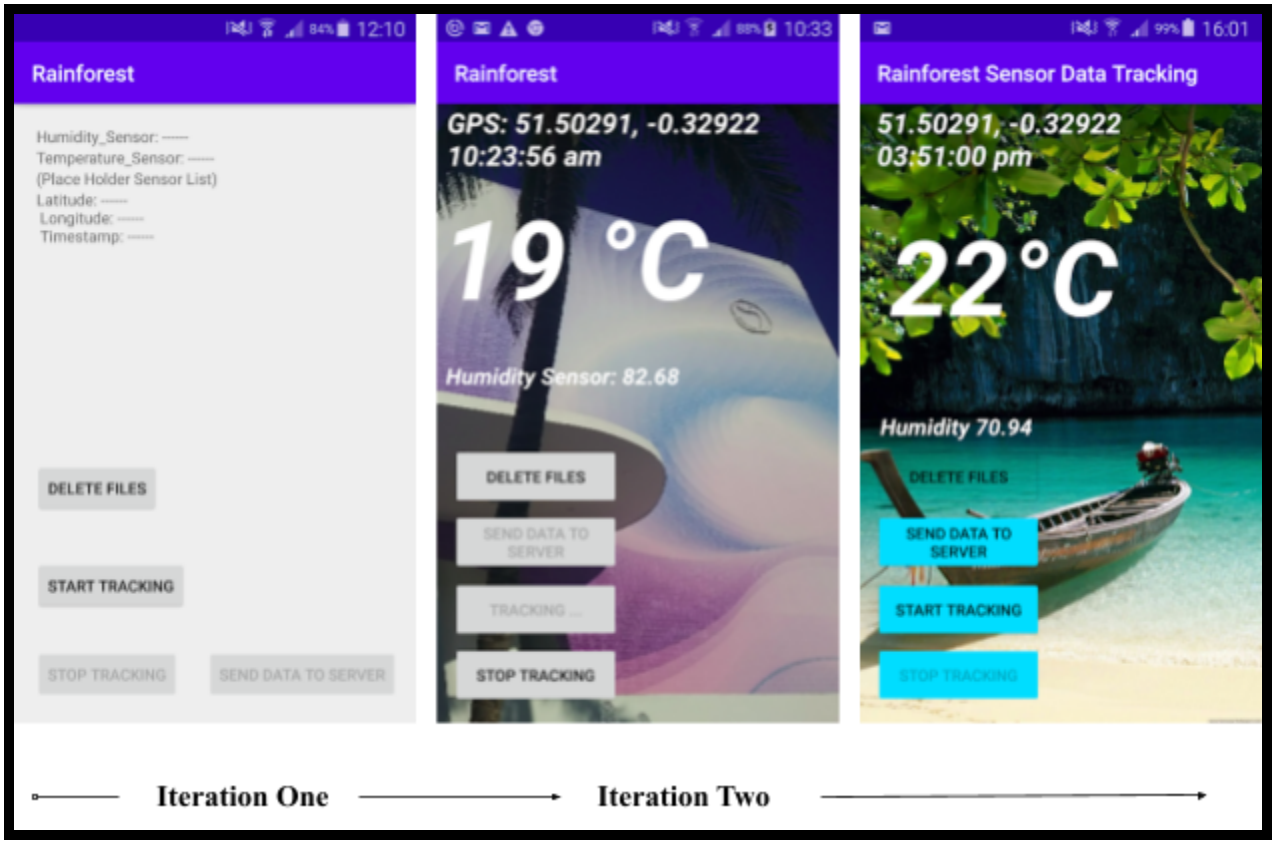


Fig. 17: Input component user interface evolution

3.5.8 Iteration One

3.5.8.0 Analysis Stage

Java was identified to be the main programming language of many Android applications formally in this phase, and due to my familiarity with Java from the degree program it was decided to write the application in Java without the use of frameworks that could increase the time it would take to implement the component and the prototype.

This attained to an additional objective of the project; to improve my understanding of Java and its application having never written an Android application but having experience with Linux; the basis of the Android operating system.

The identified hardware platform, the 'Samsung Galaxy S4 smartphone' was purchased from ebay for ~$40. The Android Studio IDE was identified to be a popular free, opensource, well reviewed and supported tool for smartphone Android application development. The remainder of the analysis phase was spent exploring the connection between the IDE and smartphone to deploy the programmed application.

3.5.8.1 Design Stage

Following the tutorials for the Android Studio IDE and other tutorials on Youtube, Google's Android developers guide and other online sources, a basic and standardised structure for Java Android applications was researched.

This basic structure was used for the Android application and the resulting Input Component artefact and is outlined below together with the core classes of an Android Application (Activity, activity_main.xml and mainActivity.java) alongside the project's specific classes detailed in the above design sections that provide the components functionality:

This basic structure and description of the design is written in retrospect of the completed component:

Core Android Application Components	**Description**
Activity	A single 'screen' of a mobile application, similar to a web page of a website, usually swiped or clicked between. This project's application has a single 'screen' for simplicity and speed of development.
activity_main.xml	This is the display structure of the 'screen'. Defining its layout, fonts and identification ID's, similar to the HTML file for the structure of a webpage.
mainActivity.java	This is the starting point of the application, the main initialising and interactivity Object of the application. Initialising the connection between the XML structure and interactivity of the Java code and its Objects.
Attribute ID's	Identify components of a 'screen', eg. TextViews and Buttons to be referenced in the Java interactivity code.

Main Project Classes

Sensors.java	=	Initialises and establishes sensor readings retrieval in a background Thread.
Tracker.java	=	Coordinates the starting, stopping and the collection of data of a 'Tracking session'.
Point.java	=	The data object for the sensor readings values, time and GPS location at 15 minute intervals for a tracking session.
Track.java	=	The data Object that represents a tracking session that stores each Point in a List.
DataStorageEngine.java		Defines the local offline storage and retrieval of the locally stored Serializable tracking session's Track objects.
TrackClient.java	=	The networking component that sends all locally stored tracking Sessions to the central server and database component.

Minor Project Classes

TrackFileNameCreator.java	A static utility class that generates a local storage file name using a Track's unique ID.

3.5.8.2 Implementation Stage

This Implementation phase followed closely the UML diagrams of the previous design phase but with additional implementation necessities and unplanned improvements to combine the design within an Android application, including the background Thread in the Sensor class and the addition of the Tracker class to coordinate the tracking session.

The above main Classes are annotated in more detail in this implementation phase section with example code extracts to clarify their implementation without having to reference the complete source code. It is worth mentioning that not all Android project code that is auto generated is included in this section (or the appendix but is available on the project's Github page) to focus on key areas.

activity_main.xml

Defines the layout structure of the GUI, with TextView and Button attributes, their IDs and some additional placeholder String values for the sensor readings.
The background image is also defined in this file but was implemented in the second iteration.

Attributes were given basic sizes, font types, colours and relative locations following the prototype interface diagram of the systems design phase but changed slightly or not implemented exactly to be open to review and to increase the speed of the implementation.

The 3 main Attribute groups are the:

- Output sensor readings
- Application functionality initiation buttons, and
- User interactivity feedback messages that communicate the internal workings of the application to the user that are demonstrated in the walkthrough video.

To illustrate, a ‘commented out’ unused extract code example is included below that was included in the first iteration to display the complete list of hardware platform sensors that are retrievable with the software code. These additional sensors are the sensors that can be implemented in future iterations.

```
<!--    for future iteration
            <TextView
                android:id="@+id/sensor_list"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="4dp"
                android:text="(Place Holder Sensor List)"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/temperature_sensor_text" />
-->
```

The ‘text view’ attribute has an ID, relative orientation, margin and placeholder text to be visible. Predefined and pre-formatted Strings for the sensor output readings and to be used as placeholders in their respective text views are defined in the strings.xml file :

```
<string name="location_text">" %1.5f, %2$.5f\n %3$tr\n"</string>
```

The above displays the GPS Latitude and Longitude in their given digits of precision for the presentation seen in the screencap above but is different to the internally recorded and stored degree of precision for the values.

MainActivity.java

The MainActivity.java class is the equivalent initialising Main class and starting point for an Android Java application with the onCreate() method being the equivalent constructor for the initialization of the application. The class imports Android, and Google classes to be available in the class.

com.google.android.gms.location

The above is imported to make available the important LocationCallback() and LocationResult() methods.

Following from the study guide and the project's objective to utilise principled software engineering principles, the MainActivity.java class and all other classes follow an Object-Oriented programming structure; importing classes, defining field variables, a constructor, instance methods, getters and setters and on optional overriding toString method.

The mainActivity.java class contains the connection to the activity_main.xml elements that are defined as field variables, as well as defining onClick action listeners for the GUI buttons that initiate and stop the core tracking functionality, networking and deletion of local data thats implementation is defined in other classes.

A key of the mainActitivy.java class is the mLocationCallback() callback method that receives periodic GPS locations at intervals defined in the Tracker object as well as creating a new Thread to read sensor readings synchronously, storing the information in a Point Object within a Track's List, and updates the User Interface of the new sensor readings, time and GPS locations.

```
            //needs to be at the bottom of onCreate to be able to make periodic location requests
        mLocationCallback = new LocationCallback() {
            @Override
            public void onLocationResult(LocationResult locationResult) {

                tracker.setmLastLocation(locationResult.getLastLocation());
                    new Sensors(context, tracker);      // starts a thread to catch the sensor values synchronously and
calls tracker to add a new Data Point
                updateUI();
                updateScreenDataValues();

            }
        };

    }
```

UI actions initiate a change of button text and toggle the disabling of buttons to restrict a logical flow of interaction with the application and to prevent undesirable side effects.

User interactions also display 'Toasts', built in Android feedback messages that are output to the user based on their action or the applications runtime errors. This was a nice built-in feature of Android development that displays on top of other xml components and without the need to define a new text field.

A nice example:

```
    private void updateUI() {
            int pointCount = tracker.getTrack().getTrackPoints().size();
            showToast(" Data Point Saved !" );
            messageHandler("Data Point: " +  pointCount  + " Saved");
        }
```

```
protected void showToast(String string) {
    Toast.makeText(context, string, Toast.LENGTH_LONG).show();
  }
```

The commented out partial implementation to retrieve and display the complete set of sensors as mentioned above is also included in the source code of this class :

```
/*
    // Get List of Sensors.
        List<Sensor> sensorList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
        StringBuilder sensorText = new StringBuilder();
        for (Sensor currentSensor : sensorList) {
           sensorText.append(currentSensor.getName()).append(
                 System.getProperty("line.separator"));
        }
        TextView sensorTextView = (TextView) findViewById(R.id.sensor_list);
```

Sensors.java

The Sensors.java class is another key class that, as mentioned above, initiates the periodic sensors reading Thread. The Thread is intended to optimise the battery life so that hardware sensors are only checked when the location call back is called.

```
mSensorHumidity = mSensorManager.getDefaultSensor(Sensor.TYPE_RELATIVE_HUMIDITY);
mSensorTemperature = mSensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);
```

mSensorManager readings (which in this iteration of the component are humidity and temperature) are retrieved from their respective hardware sensors via the imported Android hardware.Sensor class.

Some logic is implemented in the class to ensure both temperature and humidity sensor readings are retrieved before a sensor reading is returned and some @Override method definitions required by the imported Android hardware class that receive callback calls when registering for sensor readings that are updated by the operating system are also included.

Tracker.java

Tracker.java, another important class, initiates a tracking session by establishing the GPS retrieval with the imported class :

```
com.google.android.gms.location.FusedLocationProviderClient;
```

And registering with the client for location updates with the values defined in the locationRequest object that is passed to the clients constructor as an augment.

The values used for testing and for the first iteration of the prototype were:

```
private LocationRequest getLocationRequest() {

    // 5000 = 5 seconds. finalize software project requirements for sensor performance / timing
before programming.
    LocationRequest locationRequest = new LocationRequest();
    locationRequest.setInterval(900000);    // 900000 is 15 minutes, good for prototype/ testing : )
    locationRequest.setFastestInterval(900000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    return locationRequest;
```

Settings for intervals, in future iterations, could be user defined with additional implementation in the UI.

5 second intervals between GPS callbacks were used for the initial unit and integration testing and later changed to 15 minutes (900000 milliseconds) as it was considered a good starting point for field trials, system and user testing for the prototype.

High Accuracy is defined in the configuration as well but these initial values could be seen as arbitrary awaiting feedback and evaluation.

Tracker.java additionally coordinates the adding of the new data Point object to the Track object's Point List for the 'tracking session' at the defined intervals.

```
protected void addNewDataPointToTrack( float humidity, float temperature ) {

    getTrack().getTrackPoints().add( new Point(
            UUID.randomUUID(),
            getTrack().getTrack_id(),
            (float) getmLastLocation().getLatitude(),
            (float) getmLastLocation().getLongitude(),
            new Timestamp( getmLastLocation().getTime()),
            humidity,
            temperature)
        );
```

Tracker.java also coordinates the end of a tracking session when initiated by a user interaction unregistering with the locationClient.

Point.java

Point.java is the first key element in the data model that defines the data types of a Point and is stored in the List of Point objects in a single Track object for a single tracking session.

```
  public Point(UUID point_id, UUID track_id, float gps_latitude, float gps_longitude, Timestamp point_timestamp,
float point_humidity, float point_temperature) {
    this.point_id = point_id;
    this.track_id = track_id;
    this.gps_latitude = gps_latitude;
    this.gps_longitude = gps_longitude;
    this.point_timestamp = point_timestamp;
    this.point_humidity = point_humidity;
    this.point_temperature = point_temperature;
  }
```

The constructor is included to show the data types along with how the initialization of the object is implemented.
Randomly generated UUID's were initially of type int but adjusted to be generated with the :

```
UUID.randomUUID();
```

method to uniquely identify each Point following an inconsistency found in the integration testing with the central server and database component. Floats are used for the required precision of the GPS Latitude and Longitude, Timestamp for the time, and floats again for the Temperature and Humidity readings so that their accuracy may be evaluated.
Aside from getters and setters a createJsonTimeStamp() method was defined to

```
public String createJsonTimestamp() {
```

```
    String extendedTime = point_timestamp.toString() + "+00:00";
    String converted = extendedTime.replace(' ', 'T');
    return converted;
}
```

ease the networking of the Timestamp format into JSON to be sent to and consumed by the server.

Track.java

Track.java implements Serializable to make the storage and networking of the state of the object easier.
A List<> Interface was defined for an ArrayList<>() of type Point objects to store the tracking sessions Point objects. With additional String name and description fields defined, but commented out, for future iteration implementation. They were found to add complexity to the implementation as they were considered not to be specified in the primary requirements for this component.

DataStrorageEngine.java

This class coordinates the offline and local storage of the data Tracks by checking and creating if a tracks directory exists in the devices storage and using FileOutputStream writers to write a Serialized Track object to a file (1 per file) made with the TrackFileNameCreater.java class and unique Track UUID value.

The class has an additional method to check if a Track has at least 1 point before writing to the device's memory; this was discovered within the first iteration's integration testing with the server and database and implemented by the second iteration. Additional filtering at this location in the program would be beneficial to filter the data that is input into the system to be implemented in a future iteration.

All Tracks are also retrieved with this class for networking and for deletion.

TrackClient.java

This class utilises the above DataStorageEngine to retrieve and network the locally stored data to the central component of the system. The server URL address is hardcoded in the class :

```
private static String url = "http://markstart.duckdns.org:8000/tracks";
```

Within the class a method is defined to send all Track files from the device's local storage that also initiates a notification message that is displayed to the user via mainActivity.java of the softwares success to transmit the data.

```
Toast.makeText(contextClient, " Connecting to server ... ... ", Toast.LENGTH_LONG).show();
```

Every locally stored Track is sent by sending their individual Point objects, one at a time, by converting them to JSON and transmitting them using TCP and HTTP.
To implement this decision the Track UUID is also stored within each Point object.

This choice was made to increase the robustness of the networking function, where its transmission is expected to be made from remote tropical rainforest locations or from semi-isolated or patchy networks.
The intention was that the transmission of individual Points, if the communication was disrupted, if the transmission were smaller would be less likely to be disrupted but on reviewing the TCP/IP protocol this approach would need to be explored in more detail to discover if it does indeed provide any real value to the networking function of the input component of the system.

This httpClient

```
import cz.msebera.android.httpclient.Header;
```

class was imported and used as it intuitively felt to be the best implementation of HTTP for the Android application when searching online for the best way to implement this key function of the application. The process works by :

1) A StringEntity new BasicHeader object is created of content type JSON.

```
se.setContentType(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
```

2) All Tracks are retrieved from local storage to an ArrayList of Tracks, where each Point for each Track is iterated through.

```
for (Track track : tracks) {
        for (Point point : track.getTrackPoints()) {
        sendJsonPoint(point);
    }
```

3) Within the sendJsonPoint() method each Point is converted to JSON and then to a StringEntity

```
se = new StringEntity(json.toString());
```

4) Each StringEntity's contentType is set with a new BasicHeader; content type JSON.

```
se.setContentType(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
```

5) The asynchronous HTTP client is initialised.

```
AsyncHttpClient client = new AsyncHttpClient();
```

6) The client.put() method uses the hardcoded URL, the constructed payload StringEntity, and an AsynchttpResponseHandler to handle the success or failure of the communication. In the current implementation the response status codes, header and body are not processed simply communicating a message to the user that the files were sent or not sent; another future iteration improvement for refinement.

There is no control to send only a few Tracks, which could be implemented in a future iteration but from the results of the system testing and field trials this current implementation presented no problems but is formally noted in this report to be an area that could be improved.

3.5.8.3 Testing and Iteration Retrospective

Unit testing and integration testing were conducted with the server and database central component of the system following the implementation stage. The integration testing with the central component was conducted while implementing the central component to assist in its implementation as well as after its implementation phase to test their interoperability.

Admittedly formal documentation was not kept of this testing phase but important identified aspects of the first iteration that were considered to be likely to need to be addressed in the second iteration were recorded, including :

- The Android application's 'Context object' was found to be confusing and was required by Android to be present in many objects and methods, and it was felt to be incorrect to pass the Context object around so much within the program.

- The device's location information is retrieved using the 'FusedLocationProviderClient' Interface that is designed to efficiently identify the location without worrying about which provider or network is used to determine the location.
  This approach is easily customizable and optimizable by setting different parameters for different accuracy and power consumption settings. Typically greater accuracy requires greater power consumption.
  The LocationRequest object contains fields for the requirements of the location request, including:
  - Update frequency (with setInterval())
  - Accuracy (set with 'PRIORITY_HIGH_ACCURACY')
  - setFastestInterval() to limit the fastest updating of the value.

  Upon the results of the more comprehensive testing it was evaluated that this configuration was not configured to be optimised. 'Access Fine Location' user permissions were specified to retrieve the most accurate reading over the more efficient 'Access Course Location' setting that is specified in the AndroidManifest.xml file.

  ```
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  ```

  The 'getFromLocation() method was used to synchronously retrieve the GPS coordinates, but it was noticed in the documentation that this approach may take a longer time.

- Dependencies were managed with Gradle which was not considered in the analysis stage of the iteration.

- It was noted that the project's Point class has the same name as a different Java class which was not known to be technically important to change.

- UUID was needed to be used for ease of use with the rest of the system over int for unique IDs providing additional advantages so is considered an improvement to the previous implementation.

In retrospect it would have been a good idea to present the complete list of hardware sensors to the user in the UI without their complete vertical implementation to present the greater potential horizontal functionality that could be implemented in future iterations.

3.5.9 Iteration Two

Iteration two of the input component of the RIS was initiated in December after unit, integration, system and usability testing of all three components of the RIS were completed.

The iteration had less formal analysis, design, implementation and testing phases than the first and became a way to analyse and implement the improvements and aspects identified by different users asked to evaluate the component and from the results of the different approaches to testing.

A main focus of the second iteration was a reorganisation or refactoring of the structure and improvement of the modularity of the code as well as the resolution of several UI bugs that were identified in testing that were resolved.

The initial iterations source code was considerably less organised, its presentation improved and its logic clarified in this iteration.

The testing of the functionality of the component did not reveal any runtime errors but did highlight some obscurity between the internal workings of the application and the user messages that are communicated to the user. Some refinements to the functionality were made and the communication of the internal workings of the application to the user were clarified.

Some examples of the second iteration's implementation phases include the addition of the ‘connecting to server … …’ toast message that can be seen in the video walkthrough, the Point count that was added to the UI as well as the addition of the functionality that a Track cannot contain 0 Point objects before the Track is saved to the devices storage at the end of a tracking session.

Improvements to the format of some of the data that the application captures was improved to be more consistent with the formats expected in the central component, as well as some refinements to the networking class, the usability and presentation of the UI.

The background image was added and the presentation and orientation of the GUI components was adjusted to align with user feedback from user testing results.

User testing revealed that a main concern of the user of the input component was the graphical user interface. The functionality of the artefact was not nearly as critically evaluated but perhaps speaks to the ease of the automation of the design.

The testing phase of the second iteration included unit and integration testing to establish that the improvements and adjustments had not affected the functionality and interoperability of the component within the prototype RIS.

It was interesting to see the advantages of the modular and component based approach to the system architecture when implementing new features, improvements and bug fixes to the input component in this iteration.

In conclusion the second iteration was extremely useful to the development of the input component and with the reception of the final prototype system. It made possible an allocation of the project's time to address the results found in the testing activities, to better understand the program code, to improve small things and to refine the components performance. It also helped to formalise additional points that could be addressed in future iterations.

Ideally more iterations could be conducted with further improvements made to the input component with greater frequency of design, implementation, testing, evaluation and re-design.

3.6 Central Processing and Data Storage Component

The central processing, data storage database and server component was considerably faster to design, implement and test than the input component, as was the intention of the project plan. Only a single iteration was completed for both the server and database, but the additional time that was allocated for the second iteration was able to be used to resolve some of the issues that were identified in the unit, integration and system testing for this component of the system.

An analysis stage was conducted to evaluate the different approaches for the design and implementation of the central component and resulted with the artefact mostly following online tutorials for the popular framework Java SpringBoot as it was identified to be a convenient, fast and very in demand technology in the industry.

A relational PostgreSQL database was used for the data storage function within a Docker container for the central component as it was researched to be free, open source and one of the most popular approaches and databases to be used for data, specifically scientific data. The Docker container was used to increase the flexibility of deployment of the server and database but in retrospect served as more of an educational exercise than adding value to the prototype system.

3.6.0 Central Processing and Data Storage Component Design

The Server and Database central component of the client-server system architecture was designed following the well established and popular Java SpringBoot design of a 3 layered web-server, logic and database interactivity layers.

Layers include:

- A HTTP API controller layer
- A business logic, or data processing layer
- A Data Access Object (DAO) layer

The design was chosen due to familiarity, online support and to increase the speed of the implementation.

The intention was that networked Point objects from the smartphone Android application containing GPS, TimeData and sensor readings would be easy to traverse between layers and to be decomposed to be inserted into the database as well as when reconstructing, networking and mapping the Point objects of a tracking session to the web-based 3D globe interactive map.

The API controller layer would coordinate both the HTTP GET requests for the output component and the POST requests made by the input component clients.

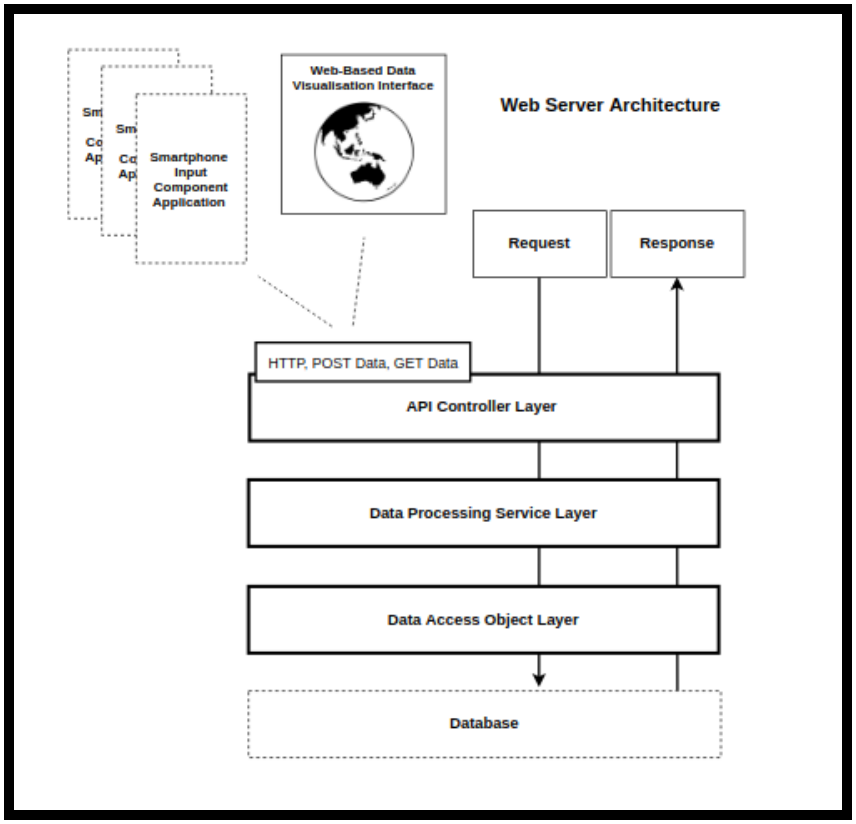


Fig. 18: Central component webserver and database architecture

A design choice was made in this initial stage to only design for these two key functions (API calls) as base requirements to support both the 'read track point sensor data use case', and the 'viewing of field data use case' resulting in an artefact that implements both a subset of horizontal and subset of the complete vertical functionality of the requirements of the prototype system.
Other HTTP requests and vertical functionality could be designed, implemented and tested in following iterations.

The data processing layer is intended to be able to process or perform calculations on the data when received or being accessed but no processing or change to that data was designed for at this stage.

The Data Access Object is designed to map the object model of the Track and Point data model to be inserted as data into relational tables with relationships in a relational database. A relational database was chosen, again due to familiarity, and for the advantages of a relational database over a noSQL database, including structure, organisation, my familiarity with SQL, consistency and other ACID properties. A common problem was identified at this stage; the Object-Relation impedance mismatch.

The design approach taken for the object-relational impedance mismatch and change between the logic of the object-oriented programming logic and the relational database logic is localised to the DAO layer to keep database logic in the database, object-oriented logic in the server and handel the conversion with the simple, light and powerful DDL (data definition language) and DML (data modification language) of SQL.

Java provides many techniques for accessing data and can automatically return retrieved data in a particular format such as JSON without explicit conversion in the code. For this implementation Spring JDBC was used with a Hikari datasource (the recommended datasource for SpringBoot) and Flyway.

3.6.1 Server

3.6.1.0 Server Implementation

Following the less comprehensive design stage (compared to the input component) the implementation of the Spring Boot RESTful API server to provide create and read (PUT and GET) functionality to the other components of the RIS followed closely online tutorials of a similar design with the additional considerations for the processing of the Track and Point data model and other project specific considerations.

SprintBoot was chosen to implement the backend server, due to my familiarity with Java from the degree program as well as being a technology that provides quick, production ready applications that are made from popular frameworks and configurations that could significantly increase the speed of deployment and provide well documented support for the application.

Using a pre existing framework for the server allowed a focus for the customisation for the project specific considerations to be implemented and to not have to write the complete server from scratch.

SpringBoot uses @ annotations and dependency injection within a Spring container that allows for changes to be made to the server easily without making lots of updates to code to make that change.

@ Annotations are used by a component-scanning function of SpringBoot to autowire (connect) objects of different parts of the application automatically. The instantiation of objects with '= new' is not used so that Spring can create and connect objects with @ annotations.

3.6.1.1 Server Configuration

Some key additional activities for configuring and implementing the server are detailed in this section as follows :

- Configuration of my home Talk-Talk router for port forwarding on local port 8000 to make the server accessible outside of my home network.
  The configuration was for TCP but could be configured for UDP in a future iteration if needed.

- DuckDNS (a free open source dynamic IP address resolution service) was installed on my Ubuntu Linux Machine through the terminal and an address/ account was registered with my name 'markstart'.
  DuckDNS runs as a daemon (background process) to continuously update the dynamic IP addresses that are issued to my router by my ISP (the ISP presumably uses a pool) so that a static address can be used to reach the web-server application on port 8000.
  This is achieved by DuckDNS by using a subdomain structure with a domain name translation service. The combined static URL address to access the server is :

http://markstart.duckdns.org:8000

- IntelliJ IDE was installed for Java Spring Boot application development as it is open source, free, popular and well supported.

- Java SpringBoot applications use a Maven dependency manager that automatically coordinates the adding and organisation of additional Java code packages for use in the application.
  A Maven POM file was initialised for the project, and updated (the process and downloads are automatically handled).

- To connect the Java Spring Boot server to the PostgreSQL database implementation a Docker container with a configured exposed port to allow access was implemented.
  Commands to start, stop and remove the Docker container used were :

  - Stop and delete all images and containers : sudo docker rm -f $(sudo docker ps -aq)

  - To run the Docker container (this command runs a container instance of the 'postgres:alpine' official Docker image (Alpine was used as it is the smallest image and contains Postgres pre-installed and configured within the image), with passed augments for the database's password and port to be exposed on the machine (-d = detache mode to run as a background process) :

    docker run --name [name of container] -e POSTGRES_PASSWORD=[passwordForSession] -d -p [expose port] postgres:alpine

    Eg. docker run –name rainforests -e POSTGRES_PASSWORD=password -d -p 5432:5432 Postgres:alpine

  - To launch an interactive session into the docker container:

    docker exec -it <container ID> bin/bash

  - To log into the Postgres database with the Postgres user defined in the configuration file.

    psql -U postgres

3.6.1.2 Server Application Code

Within the 3 layers of the application, key classes are described in this section. The server applications directory structure organises the three layers in to directories 'api', 'model', 'dao' with additional 'model' and 'service' directories and an additional class RainforestApplication.java class that contains the initiating main method of the application with the accompanying @SpringBootApplication annotation.

Java classes follow the same principled structure used in the input component classes with the addition of SpringBoot techniques. Only key aspects are focused on in detail in this section.

Track and Point Classes

Within the central layer, and the server's data 'model' directory (a standardised location and naming convention), corresponding Track and Point Classes are defined with @JsonProperty annotations to be able to reconstruct and networked the JSON data transmissions when networked from the input component and to the output presentation component of the RIS.
The @JsonProperty annotation makes the JSON format consumable as the media application type at the API layer (detailed in the next section) for these data model classes.

```
public Point(@JsonProperty("point_id") UUID point_id,
        @JsonProperty("track_id") UUID track_id,
        @JsonProperty("gps_latitude") float gps_latitude,
        @JsonProperty("gps_longitude") float gps_longitude,
        @JsonProperty("point_timestamp") Timestamp point_timestamp,
        @JsonProperty("point_humidity") float point_humidity,
        @JsonProperty("point_temperature") float point_temperature) {
```

PointController.java

As the top API layer providing REST services with the @RestController annotation, although only create and read (PUT and GET) functionality is implemented, the PointController.java class is surprisingly small, only ~40 lines long, showing the great advantage of using SpringBoot to implement the server for the prototype RIS.

The @RequestMapping("/tracks") annotation and argument defines the URL location for the REST services. Where all requests made to the server use this base address URL :

http://markstart.duckdns.org:8000/tracks

Additionally the @CrossOrigin annotation was needed to be able to test the server from the local host with the output component but not with the input component.

```
@CrossOrigin        //needed for localhost connection from html/.js client
@GetMapping
public List<Track> getAllTracks() {
   return pointService.getAllTracks();
}
```

The @GetMapping annotation is defined to handle GET requests. GET requests made to the above URL will call the getAllTracks() method in turn calling the PointService.getAllTracks() method returning a List of all the Track objects retrieved from the database.

An individual Track's List of Points can be retrieved from the API with its unique ID with the defined :

@GetMapping(path =/{"track_id}")

From the address : http://markstart.duckdns.org:8000/tracks/[track_id]
The URL location in turn calls pointService.getAllPoints(track_id).

```
   @RequestMapping(method = RequestMethod.PUT, consumes =
MediaType.APPLICATION_JSON_VALUE)
   public void addPoint(@RequestBody Point point) {
      pointService.addPoint(point);
   }
```

Finally, the above code coordinates the PUT requests for the individual Point JSON payloads. The payload is mapped with the @RequestBody annotation corresponding to the annotation in the Point Object.
As mentioned earlier a design decision was made to network individual Point objects to increase the reliability of the networking function of the RIS that could be disrupted in patchy network areas in remote locations.

PointService.java

The PointService Class is the middle data processing layer of the server connecting the API layer and the Database Access Object layer and contains the implementation of the above called methods.
No additional data processing was implemented in the prototype but this layer is defined in a way that it can be easily extended in this way.
The layer was mostly used to understand and debug the networking of the data at different stages in system :

```
System.out.println("INSERT INTO points VALUES ('"+ point.getPoint_id() + "','" + point.getTrack_id() +"','"+
                point.getGps_latitude() + "," + point.getGps_longitude() + ",'"+point.getPoint_timestamp()+"');" );
```

Methods are called and in turn call the database access layer methods passing List objects of Track and Point objects between the three layers.

PointDao.java

PointDao.java defines an Interface defining the methods to be implemented in the DAO layer to allow an easier change of implementation of database interactivity for testing and for future iterations. The database, server, or both could potentially be hosted on a cloud computing platform. The PointDao.java interface together with the applications configuration file can be used to quickly and easily change the connection between the server and a database.

Methods defined with their arguments and return types include :

1. void insertPoint(<Point values>);
2. List<Track> selectAllTracks();
3. List<Point> selectAllPoints(<Track_id>);

PointDataAccessService.java

This Class is the main class in the Data Access Layer and is annotated as an @Repositry("postgres") for SpringBoot to be able to Autowire the connection between the Java layer and the database with the JdbcTemplate object able to coordinate and simplify the coordination, connection and queries made to the database, including the handling of some common SQL and runtime errors.

The three methods in the above layer and interface are implemented in this class with String defined SQL statements with their values passed from the method calls and together with the JdbcTemplate object send data to and from the database.

```
if (pointInDB(point_id) == false) {
    jdbcTemplate.update(
        "INSERT INTO points VALUES ( ?, ?, ?, ?, ?)",
        point_id, track_id, gps_latitude, gps_longitude, point_timestamp
    );
```

Returned data from the database is retrieved in a similar way with String defined SQL and the JdbcTemplate in a ResultSet.

```
@Override
public List<Track> selectAllTracks() {
    final String sql = "SELECT track_id, track_name, track_description " +
            "FROM tracks";
    return jdbcTemplate.query(sql, (resultSet, i) -> {
        UUID track_id = (java.util.UUID) resultSet.getObject("track_id");
        String track_name = resultSet.getString("track_name");
        String track_description = resultSet.getString("track_description");
        return new Track(track_id, track_name, track_description);
    });
}
```

To reconstruct the Track and Point objects, different queries are made to different tables :

```
        float point_humidity = getPointHumidity(point_id);
        float point_temperature = getPointTemperature(point_id);

        return new Point(point_id, track_id, gps_latitude, gps_longitude, point_timestamp,
point_humidity, point_temperature);
    });

}

private float getPointHumidity(UUID point_id) {
    final String sql = "SELECT sensor_reading FROM point_sensor_readings WHERE point_id = '"
+ point_id + "' AND sensor_name LIKE 'Humidity'";
    return jdbcTemplate.queryForObject(sql, Float.class);
}
```

Additional error handling code was written to check that Point and Track entries have not been entered into the database with the same ID and to not conflict with Primary Key constraints. Since this implementation I have learnt it is better to filter data at different stages before the data reaches the DAO layer.

In addition to the above classes there are some additional database configuration files that are detailed in the appendix and on the project's Github page. The Server implementation was quick and simple, with most of the complexity being focused in this data access layer.

3.6.1.3 Server Unit and Integration Testing

Unit testing of their server component revealed a need for additional error handling in the DAO layer that was implemented within the same iteration. It was found that it appeared that JDBC was batch inserting data into the database. This necessitated extra error handling code to handle duplicate inserts that were causing the server but not Postgres to crash.

Integration Testing between the central and output components of the system revealed problems with CORS cross origin resources that necessitated a simple SpringBoot @ annotation change in server's API layer.

Integration testing also revealed a mismatch in the formatting of the time and dates between the different components of the system that also necessitated a change in the input component and the presentation of the time and date in the output component.

It was also identified that very limited response code handling, simplified to booleans being used to indicate the success of the data entry event into the database, was adequate but noted to be an aspect that could be improved in a future iteration.

Overall the server implementation was a very streamlined and easy process with only a few bugs being discovered from the components unit, integration and the results of the complete system's testing.

3.6.2 Database

3.6.2.0 Relational Database Design

The process for the conceptual and logical design of the implementable model of the data storage component of the prototype RIS followed a very procedural approach for making a relational database.

The Design of the database was preemptively decided to be a Relational database due to my familiarity from the degree program and familiarity with SQL and that relational databases are considered very consistent, stable and have excellent data integrity.

The analysis phase for the data structure and the database utility followed closely the requirements defined in the requirements specification section of the project and report.

The development process followed closely the diagram and process included on the next page (taken from the subject guide) to incorporate both the Track and Point data model as well with additional real life contextual information about the tropical rainforests of the world.

The summarised procedural and formulaic process stages included:

1. Modelling the real-life rainforest data and Track and Point data model into an E/R (entity relationship) diagram.
2. Refining or transforming the E/R diagram for compatibility with a relational database.
3. Creating a relational model.
4. The relational model was then normalised to improve its consistency, reduce update, insertion and deletion errors and to improve the long term correctness of the data.

There was no real need to revisit earlier stages or a formal second iteration and the design and implementation process proved to be quite simple following the subject guide taking on a more waterfall process.

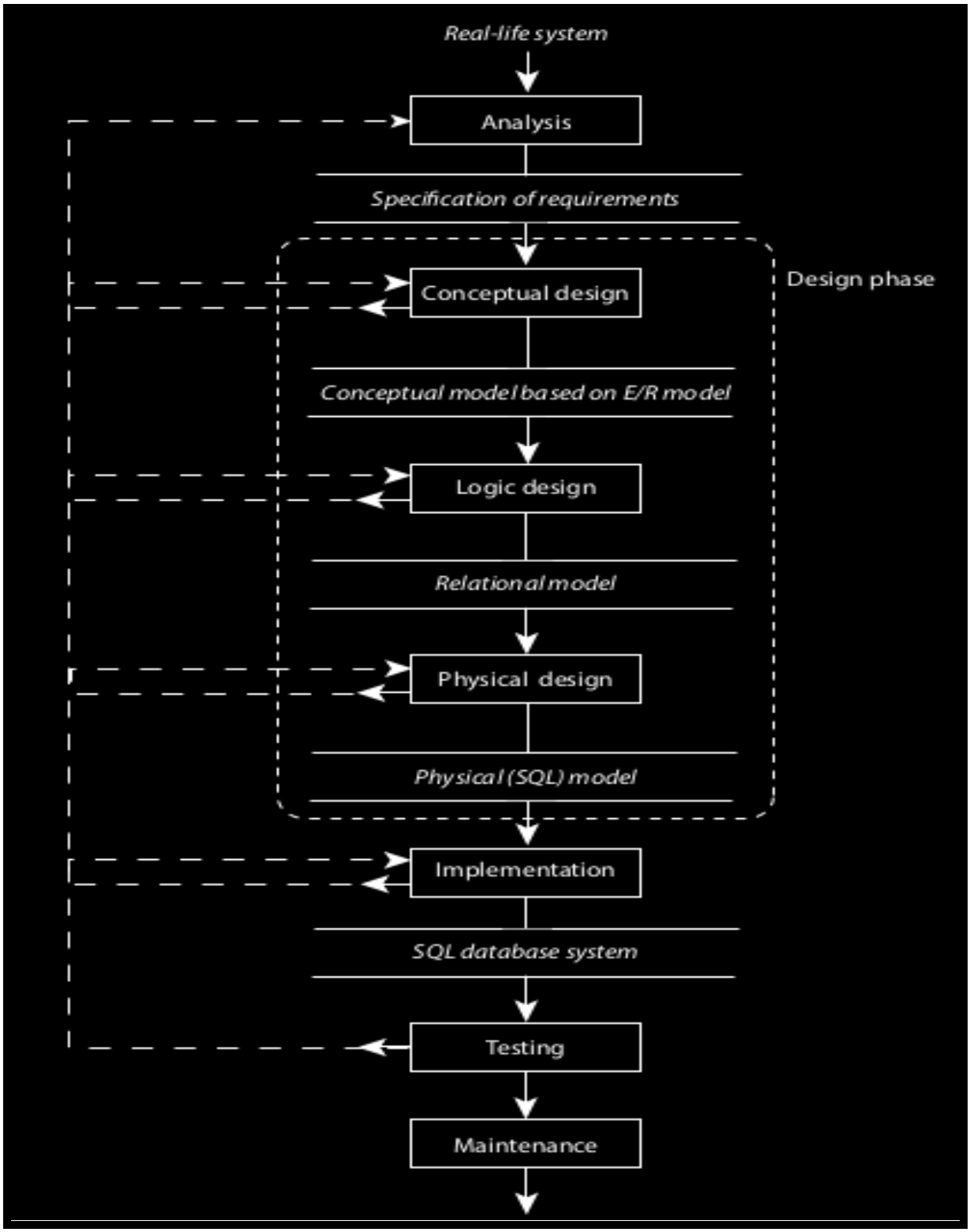


Fig. 19: Database systems development process (Lewis, 2016, p.105).

3.6.2.1 Modelling for the Conceptual E/R Diagram

Entities were identified with their attributes, relationships, constraints and their hierarchy. Entities uniquely identify a 'thing' to be modelled with their attributes defining additional information about the entities that were identified.

Simple atomic attributes were used to correspond to base attributes for relational compatibility and first normal form normalisation purposes.

Relationships connecting entities were defined, with both weak and strong existence (dependent upon the existence of the other entities in the database) and both partial and total relationships considered.

Relationship's min and max were defined to avoid fan traps and a hierarchy of entities was structured for Regions, Subregions, Forests, Tracks and Points entities to avoid chasm traps.

3.6.2.2 Refining the E/R Diagram

Transforming the preliminary E/R diagram to an E/R diagram that only uses compatible relational concepts was used to ensure that:

- Primary key attributes (primary keys are defined to uniquely identify any instance of the entity).
- Flatten and have no compound, multivalued for calculated attributes (e.g Latitude and Longitude, instead of GPS).
- To define new relationships between entities where a ternary relationship existed (e.g Forest, Koppen Type and Track) to simple binary relationships ensuring that they couldn't have attributes and only connecting at most 2 entitles.
- Simplifying relationships with no (n;m) relationships.
- Ensuring weak entities have foreign keys to strong entities.

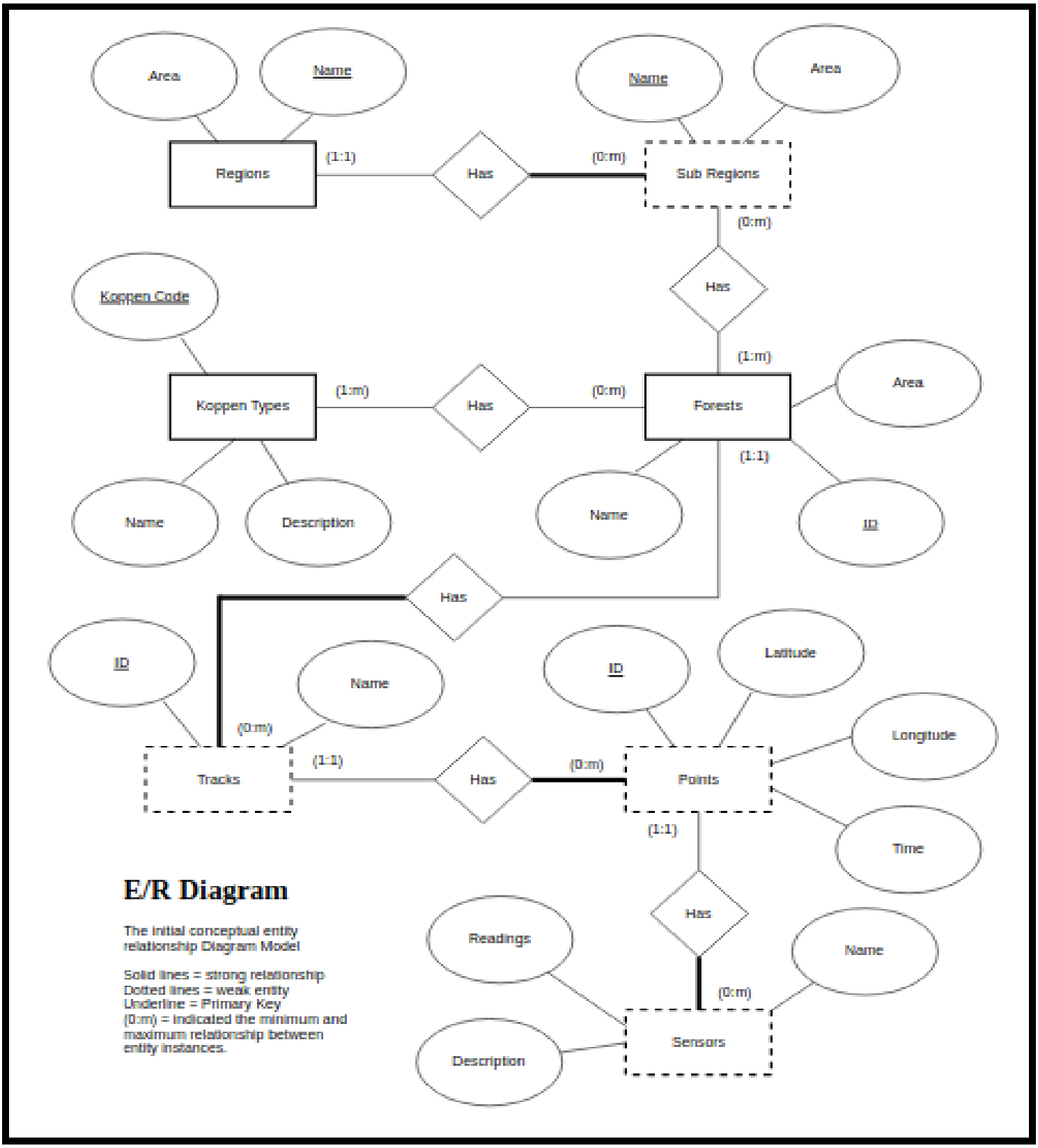


Fig. 20: Refined Entity-Relationship diagram.

3.6.2.3 Transforming the E/R Diagram to a Relational Model

The above E/R diagram was then used to create the following relational model.

Entities with relationships have another entity's primary key as a connecting foreign key to represent the relationship. If both the entities' relationship was partial, or total, then the decision was arbitrary, and where one relationship was strong, that entity would contain the forign key for the partial relation.

The below relational model was ultimately used to program the DDL and test data SQL used with Flyway within the server (described in the next section). Each entity has a primary key, attributes and an optional foreign key.

Track and Point ID's naturally correspond to relational database table primary keys. Regions, Subregions, Koppen Types and Forest tables were used to contextualise the collected tracking session sensor readings.

<u>REGIONS</u>
+ region_name <pk>
+ area

<u>SUB_REGIONS</u>
+ sub_region_name <pk>
+ area
+ region_name <fk>

<u>FORESTS</u>
+ forest_id <pk>
+ forest_name
+ area
+ forect_description

<u>FORESTS_SUB_REGIONS</u>
+ forst_id <pk> <fk>
+ sub_region_name <pk> <fk>

<u>KOPPEN_TYPES</u>
+ koppen_code <pk>
+ koppen_name
+ koppen_description

<u>FORESTS_KOPPEN_TYPES</u>
+ koppen_code <pk> <fk>
+ forest_id <pk> <fk>

<u>TRACKS</u>
+ track_id <pk>
+ track_name

<u>TRACKS_FORESTS</u>
+ forest_id <pk> <fk>

+ track_id	<pk>	<fk>

<u>POINTS</u>

+ point_id	<pk>	
+ track_id		<fk>
+ gps_latitude		
+ gps_longitude		
+ point_timestamp		

<u>SENSORS</u>

+ sensor_name	<pk>	
+ description		

<u>SENSOR_READINGS</u>

+ point_id	<pk>	<fk>
+ sensor_name	<pk>	<fk>
+ value		

3.6.2.4 Normalisation to BCNF 3.5 Normal Form

By identifying and removing functional dependencies that connect attributes in undesirable ways, redundant data that would otherwise increase the complexity of updating operations that would result in more likely errors, inconsistencies, update, insertions and modification anomalies can also be removed from the database.

By following progresssive 1NF, 2NF, 3NF, BCNF normal form rules that are inclusive of previous normal forms a more robust design was achieved. A non lossy decomposition of the relations was found to be possible.

- 1NF ensures that all attribute values are atomic.
- 2NF, that every non-key attribute is irreducibly dependent on the Primary Key.
  When (A -> B) where before (A,C -> B) (A being the determinant, and B the dependent), and, if and only if for an A there is only one B.
- 3NF ensures there are no transitive functional dependencies (A -> B -> C) == (A -> C).
- BCNF every functional dependency determinant is a candidate key.

Rule of thumb rules were also used including that:

- Each entity should map to a single entity in the real life model.
- Relations should not store information for 2 distinct entities.
- All information of a real world entity is not kept in a single entity.
- In general relations were decomposed to the fewest new entities that are in BCNF normal form.
- Relations were kept independent to avoid unnecessary consideration or risk in the relationship.

3.6.2.5 Koppen Climate Classification Types

The Koppen climate classification system categorises different geographical climate groupings that are further divided based on precipitation and temperature.

For the Rainforest Information System the tropical group ‘A’ and it’s subcategories ‘Af’, (Tropical Rainforest Climate), ‘Am’ (Tropical Monsoon Climate) and ‘Aw/As’ (Tropical Savanna Climate) are included in the database to provide additional contextual information for the Track and Point data retrieved. In further iterations of the system it would be possible to add additional complementary rainforest information for additional functionality, presentation and comparison and visualisation in the output presentation component of the RIS.

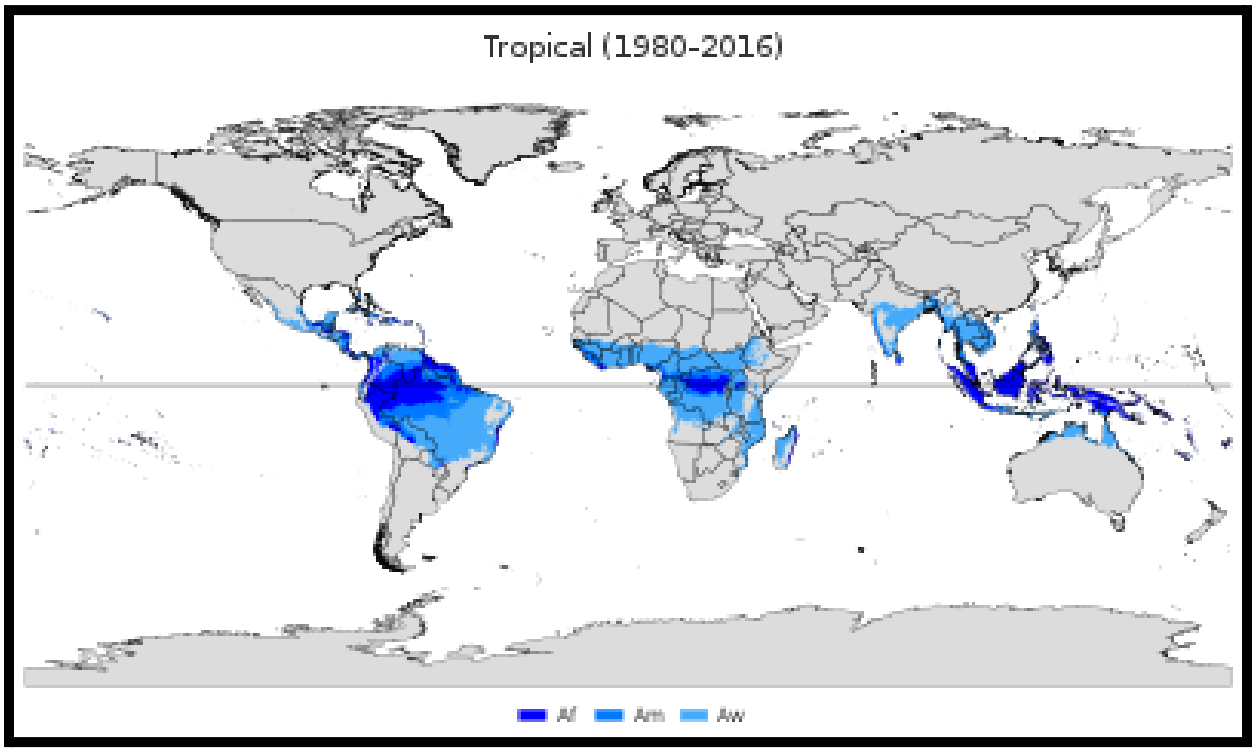


Fig. 21: Koppen Climate groups, Af (Rainforest), Am (Monsoon), Aw/As (Savanna, Dry winter/summer), (creative commons).

3.6.2.6 Database Implementation

An overview of the Server and Database are available at the youtube walkthrough:

https://www.youtube.com/watch?v=kujqOj48kVg&t=168s

The SQL DDL (data definition language) is located in the server application at in the location :

RainforestServer/src/main/resources/db/migration/V1__PointTable.sql

with the corresponding database configuration file ‘application.yml’ in the same resources directory defining the external database configuration values.
The ‘V1__’ portion of the file name corresponds to the versioning of the SQL used by Flyway (a database migration tool that went largely underutilised).

The SpringBoot server and the Docker container coordinate the initialization of the database within the containerized Postgres instances.

The JDBC URL identifies the type and location of the database to the server.
In the Jdbc URL the database is located on the localhost at port 5432 that is the mapped port from the way the container is instantiated with the Docker command defined in the Server Implementation section of the report.
The database is named 'rainforests' within Postgres at the above location that is accessed with the username and password details in the configuration file illustrated below:

```
app:
    datasource:
    jdbc-url: jdbc:postgresql://localhost:5432/rainforests
    username: postgres
    password: password
    pool-size: 30
```

The complete SQL DDL that closely follows the normalised relational model and SQL test data are available in the Appendix of the report and on the project's Github page.

The Forests_Sub_Region table and the Points table are included in this section for examples:

```
CREATE TABLE forests_sub_regions (
   forest_id UUID NOT NULL,
   sub_region_name VARCHAR(100) NOT NULL,
   PRIMARY KEY (forest_id, sub_region_name),
   FOREIGN KEY (forest_id) REFERENCES forests
      ON DELETE CASCADE,
   FOREIGN KEY (sub_region_name) REFERENCES sub_regions
      ON DELETE CASCADE
);
```

The SQL closely follows from the normalised design with additional database considerations e.g. 'ON DELETE CASCADE' and 'NOT NULL' to enforce the relationship between entities.

```
CREATE TABLE points (
   point_id UUID NOT NULL PRIMARY KEY,
   track_id UUID NOT NULL,
   gps_latitude DECIMAL(10, 7),
   gps_longitude DECIMAL(10, 7),
   point_timestamp TIMESTAMP NOT NULL,
   FOREIGN KEY (track_id) REFERENCES tracks
      ON DELETE CASCADE
);
```

The above SQL shows the actual implementation of the degree of the GPS accuracy possible.

Some initial DML (data modification language) SQL test data that is used to populate the database is also defined in the above V1__PointTable.sql file. The test data populates tables Regions, Subregions and Koppen Types with meaningful real world contextual information

Some examples from the test data include :

```
insert into koppen_types values ('Af','Tropical Rainforest',' Usually found within 10 to 15 degrees latitude of the equator,
and has at least 60 mm of rainfall every month.');
insert into regions values ('Oceania', 8525989);
insert into sub_regions values ('New Zealand', 268021, 'Oceania');
```

```
INSERT INTO tracks (track_id) VALUES ('81b0eae3-e9b1-4c6b-95b7-88c99f630aa0');
INSERT INTO points VALUES
('5bff4138-d43f-4ebc-9399-702e3d02820f','bc6307e7-1ef4-43df-8bfb-5d50735206d9',51.502907,-0.3292215,'2020-10-27 11:48:46.686');

INSERT INTO point_sensor_readings VALUES ('4564d42b-ec25-46c6-8978-c7480478d352', 'Humidity', 79.04221);
INSERT INTO point_sensor_readings VALUES ('4564d42b-ec25-46c6-8978-c7480478d352', 'Temperature',
13.599094);
```

The database implementation in comparison to the design of the database was very quick with most of the logic and structure having already been defined by the design. It is recommended to view the youtube walkthrough so see a quick overview of the tables, structure and interactivity of the Database within Postgres within the Docker container using SQL from the CLI terminal interface.

The video walkthrough also captures the type of unit, integration and system testing that was conducted on the database and server. Very few issues were identified that could not be immediately addressed in the current iteration of the central component.

3.6.2.7 Database Testing and Retrospective

Following the implementation and testing phases for the central component there were a few considerations that are recounted below but overall the implementation, testing and almost waterfall process of the database iteration produced a very stable, reliable and robust digital artefact.

Identified considerations with the database component :

- There was a conceptual problem of how to subcategorize continents without using countries, as countries change and are independent of physical geography. It was discovered that the United Nations geoscheme uses Regions and SubRegions instead of potential flexible country borders and this approach was used for sub regions within continents.

- The relational data model does not account for Point data taken over oceans. GPS coordinates can be logged but no ocean table is defined within the database. Similarly sensor data can be collected at all Points but has limited context outside of a forest.

- It was thought that a Track could have an extended relationship for different types of Track.

- It proved tricky to store different sensor data types sucinenclty in one relational table.

- A forest can exist in the database without a subregion.

- A mix of surrogate and natural primary keys were used which has not shown any problems so far. Id’s were used where real word conditions or events could cause redundant data or result in a less correct database.

In conclusion it was considered a very logical choice to contain the database logic in the database and separate object-oriented logic in the server. This allows for two separate components that have low

coupling and high cohesion. As an example the database can be accessed and interacted with from the command line without consideration for the server.

The database component's design and implementation phases were well considered but the small volumes of data are not substantial enough to evaluate the consistency of its structure. However the results of the component testing and evaluation point to a component that is expected to perform well for the purposes of the prototype system and beyond.

3.7 Output Presentation Component

The design for the output component follows from the requirements specification that necessitate a web-based interactive presentation of the RIS's data where subsets of the data can be visualised on a map or 3D globe.

Additionally :

- The mapping of the Track and Point data should include the GPS, sensor readings, time and data, and incorporate additional contextual tropical rainforest information.
- The map should have different overlays including satellite and have different zoom levels.

The use case for the 'view track point sensor data' for the identified user task of the output component, as before, could also be refined at this stage.

View Track Point Sensor Data
Goal Level : Level 2
Main Success Scenario :
1. User (Agent) loads map/globe
2. User selects Track of Points to display
3. User can zoom in and out and display different map overlays
4. User can reset the displayed Information

Extensions
2.a User can select a subset of the data; an individual Point to display.

It was decided to use Javascript for the implementation of the front end component as I have some familiarity and experience with JavaScript and it is known to be one of the main front end web-based programming languages.

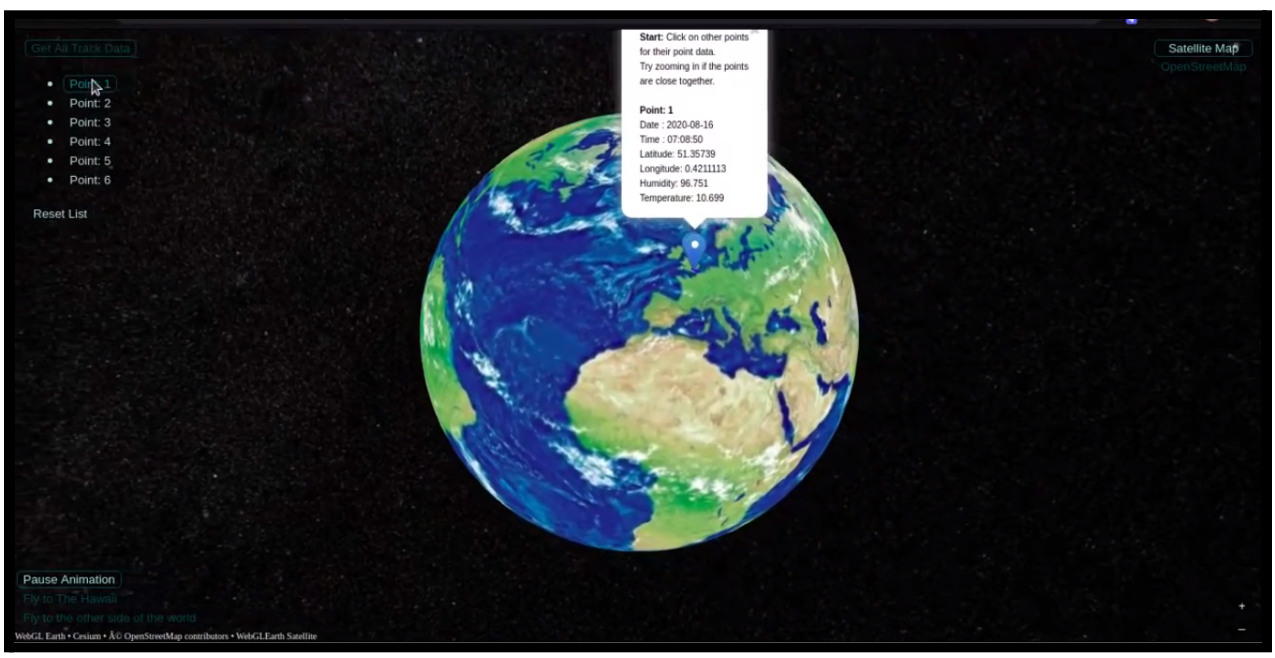


Fig. 22: Output presentation component

3.7.0 Web GL Earth

The front end presentation component mostly followed the online WebGlEarth tutorial:

https://examples.webglearth.com/#markers

Which conveniently was found to provide all of the requirements specified for the component. The tutorial was adapted and built upon to contain the project specific functionality of the component and the functionality requirements of the prototype RIS.

WebGlEarth is a free, open source (under the Apache-2.0 Licence) 3D interactive globe that can run in any browser. That can visualise data points and location information, draw polylines between points, provide different open source licence free map overlays (satellite and road maps) and can zoom into and animate the 3D globe.

GPS coordinates plotted with map markers can contain additional data and information for the marker and be connected with polylines, almost the exact functionality needed to plot the Track and Point data model onto a 3D map.

The tutorial covered many of the requirements for the output component with only the main HTTP interaction with the webserver to retrieve the Track and Point objects needed to integrate the web page component with the rest of the system.

It was planned to use Javascript to retrieve the JSON Track and Point objects and the existing WebGLEarth functionality to present them.

WebGLEarth is similar to LeafletJS, which is the 2D mapping tool that is used by the Rainforest Connection organisation in their privately facing online web based RIS. I was aware of WebGlEarth before my internship with Rainforest Connection but was pleasantly surprised to see the similarity in approach between my project and the more established RFCX RIS.

The : ‘© Copyright 2014-2018 Klokan Technologies GmbH webgl tutorial code’

text was maintained in the input component to show the basis of the implementation, with my own project specific additional code and functionality described as follows.

3.7.1 Analysis, Design and Retrospective Stages

The analysis and design for this component was considered to be sufficiently detailed in the initial system architecture and overlapped with previous design stages in the project.

When combined with the WebGLEarth tutorial, the already implemented web-server API and the IS architecture structure of the project, the plan for the output component was considered to be well defined without the need for a formal graphical design representation.

This final output component would complete the IS feedback loop for all three major components of the system. Making it possible to network data between the components and to present the retrieved sensor data information to a user of the system.

A more heuristic analysis determined that the design of each previous stage had enough detail, in the right places to avoid the need to design and document every aspect of this component with the same attention to detail, with the intention to increase speed and without any perceived loss of value from the final implementation and the final project report.

There was an expectation that future iterations could encompass their own design stages, concluding that an alignment and re-design of the component to user feedback, test results, the IS feedback loop and the prototypes evaluation would provide more meaningful source material for a more effective re-design.

The design for this component (particularly the UI), however, could have been completed more comprehensively in retrospect; the design phase met its purpose within the time allotted for the task and the tutorial used increased the speed of the completion of this activity.

Unit and integration testing and finally system testing for the complete RIS revealed minimal errors with only small adjustments needing to be made. The main shortcoming of this component is that it was not found possible to draw disconnected polylines between GPS points on the globe with WebGlEarth polylines needing to connect at least 2 GPS points.

The retrieval of supporting contextual tropical rainforest information from the database was also limited by the API of the server. And in future iterations it would also be possible to improve the presentation of the Point data within the GPS point markers.

I was really pleased with how the 3D globe turned out, but realize that it could be a single presentation component within a web page of many data visualisation components, and look forward to looking into this in the future.

Some additional programming following the testing phase is described at the end of the following section.

3.7.2 Implementation

Following the design and WebGLEarth tutorial a simple 3 file web page located in the static resources folder of the web server was programmed. A HTML file outlining the structure, a CSS file defining the presentation and a vanilla .js Javascript file containing the 3D globe and the interactivity of the webpage.

The three pages are described in more detail in this section.

rainforestGlobe.html

A simple <head> section defines the title, name, location of the .css and .js files and most importantly the <script src=> tag for the location of the WebGLEarth API.
The following <body> section defines (division) tags for the ‘map’ and it’s buttons including, ‘zoomButtons’ and ‘animationButtons’ and a division for the TrackDataContainer that displays and provides the interactive menu for selecting different Track and corresponding Points to be displayed on the 3D globe.

```
<div id="zoomButtons">
 <input type="button" class="transparentButton" value="+" onclick="zoomInOne()"><br>
 <input type="button" class="transparentButton" value="_" onclick="zoomOutOne()"><br>
</div>
```

This above code extract defines the zoomButtons division that are of the transparentButton class presentation defined in the referenced .css file and fires the zoomInOne() function event when the button is clicked that is defined in the referenced .js file as an example.

```
<div id="trackDataContainer" >
  <input type="button" class="toggleButton" value="Get All Track Data" id="getdata" onclick="getTrackData()"><br>
  <div id="trackdata"></div>
</div>
```

This important button initiates the getTrackData() function.

rainforestGlobe.css

This presentation file for the web page sets the colour theme and locations of the different buttons. Points to note include the #map division is given a position attribute of absolute !impotant; and the #trackDataContainer and other buttons are given the z-index: 100000; value to ensure they are presented at the correct position when the webpage is rendered. The complete .css file is available in the appendix and on the project Github page but is best experienced from the youtube walkthrough video.

https://www.youtube.com/watch?v=0TvVNaYX_e0&ab_channel=markstart00

rainforestGlobe.js

This file contains the main interactivity of the webpage.
Some variables are defined for the map and animation of the globe and a constant URL for the location of the server: http://markstart.duckdns.org:8000/tracks. (currently offline).

The init() function first called by 'onload' at the start of the webpage in the <body> tag in the .html file creates the map from the WebGLEarth referenced API with some arguments passed to generate the desired type and starting orientation of the 3D globe.
The applySatelliteMap() and performAnimation() functions are also called.
ApplySatelliteMap() sets the tileLayer for the map (an open source satellite map layer taken from the WebGLEarth tutorial) and toggles corresponding HTML buttons to indicate this tile layer is set.
A similar applyStreetMap() function is also defined.

Code to pause the rotating animation of the globe is defined and proved quite tricky and a thankful reference to a helpful stackoverflow post is included in a comment that makes use of the above animation variables.

Some buttonToggling code is defined with important functions such as flyToHawaii() and antipode() (the inverse coordinates of the current location, that is not used in the webpage).

The more important getTrackData() function fires a fetch GET request to the above defined URL and handels the response. It is important to note the error handling of this function is not programmed comprehensively.
The JSON successfully received from the URL is the complete list of all Tracks that are then passed to the displayTrackList() function that creates a and an ordered list for each item in the list that is then presented to the user.
An additional 'reset' button is added to the TrackDataContainer to reset the menu selection.
Each Track item list element is displayed as a clickable link that fires the getPointsData(<track_id>) function that in turn makes a GET request for all Points for that Track from the server and database at location 'URL + /trackId' of the @RestfulService API of the server.

The returned JSON is displayed by the displayPointsList() function in the same way as the displayTrackList() function with the addition that each Point is also displayed as a 'marker' on the globe containing the data of the Point.

```
marker.bindPopup("<span style='font-size:10px;color:black'>" +
  "<b>Start: </b>Click on other points for their point data.<br> Try zooming in if the points are close
    together.<br><br>" +
  "<b>Point: " + [i + 1] + "</b>" +
  "<br>Date : " + pointDateTime[0] +
  "<br>Time : " + pointDateTime[1].split('.', '1') +
  "<br>Latitude: " + item.gps_latitude +
  "<br>Longitude: " + item.gps_longitude +
  "<br> Humidity: " + item.point_humidity +
  "<br>Temperature: " + item.point_temperature +
  "</span>", options).openPopup();
```

Finally the drawLinesBetweenPoints(<json list of points>) function connects each drawn marker on the globe with the GPS Latitude and Longitude contained in each Point object.

Each Point in the menu list is clickable to pan directly to that location on the globe, and together, with the reset functionality, provide the user with the ability to clear the current markers and show a different Track's location Points and the collected sensor readings data of that Track.

2.7.3 Web Page Summary

The webpage's implementation provides the user with the core functionality of the original and the refined 'view data' scenario and use case. A user can :

- Retrieve and select individual Tracks and Points of collected data.
- Visualise the data on a navigable, interactive and zoomable map.
- The 3D globes map tile layers can be changed between satellite and street map overlays.
- All Track data is retrievable from the server.
- The displayed data can be reset.

Following unit, integration, system and user testing some additional functionality was implemented to improve the usability and refine some technical bugs in the web page and some consideration was included to improve the presentation and navigation of the pages functions.
Including:

- Track UUID's remained but Point Ids were replaced with numbers (eg. Point 1, Point 2 …) to make their navigation and display more natural.
- When a Point is selected it is automatically located on the map at a reorienting zoom level and angel orientation.
- The data and time of the Points are presented separately to be more clear from their JSON and internally stored format.
- Zoom buttons were added in case the user does not have a scrollable mouse.
- Some cool 'Fly to the other side of the world' functionality and the MilkyWay background was included to show the potential range of the prototype.

3.8 HTTP, TCP and Satellite Communication

The HTTP and TCP protocols were chosen to network data between the components of the system to increase the speed of the prototype's development. It was planned, and is part of the primary requirements to deploy the system on hardware that has the ability to use alternate networks and protocols such as the GSM network and Bluetooth to transmit data from the field to the central component of the RIS.

I was really interested, additionally researched, and hoped to attempt to network data through satellite communication to increase the range of the system for more remote locations but this approach was found to be to great for the limited scope for this project, but it was exciting to work with GPS data and satellite imagery and to map the Track and Point data model on to a 3D globe from the perspective of a satellite.

During my internship with Rainforest Connection it was discussed how a topology of recycled smart devices and a satellite component could network data from remote tropical rainforest locations to a cloud based server.

HTTP and TCP were chosen at the analysis stages as popular, logical and reliable choices to communicate retrieved field data from the input component to the web-server component and from the server to the output presentation component.

3.9 Prototype RIS Hardware

The recycled smart device hardware for an in-the-field prototype system were re-used from a previous interest of mine so are not reported in as much detail and perhaps should also be considered out of scope for the project but are included in this section to show that the prototype system was designed and implemented to be used as a complete in-the-field Rainforest Information System.

Fig. 23: Prototype RIS hardware

The above hardware was informally tested in tropical rainforest conditions in Central and South America (additional waterproofing was simply implemented by keeping the hardware in ziplock plastic bags with humidity Silica gel sachets inside). The hardware performed excellently and without errors and was considered to be a well tested component for use as the hardware for the software of this project.

The key requirements of the hardware is that it is operational in high temperature and high humidity tropical rainforest conditions, is low cost, recycled, portable, durable, rugged, lightweight with reasonable processing power and has optional additional weather and insect proofing.

In general the server and database software can run on any device that can connect to the Internet, make available the 8000 port, install and run Docker and launch the compiled Java .jar from the command line.

Following this post : https://hackaday.com/2016/11/05/turn-that-old-tablet-into-a-sub-100-linux-laptop/ from 2016. It was possible to convert a recycled 2012 Google Nexus tablet into a mini Linux laptop with a full Ubuntu desktop distribution of Linux installed with a Bluetooth keyboard, mouse trackpad and case for around ~$60 from ebay and Amazon.

The recycled mini Linux laptop supports the free and open source Firefox web browser that can easily render the output presentation component 3D globe and with the full desktop Linux terminal the Java 1.8 JRE and Docker were easily installed from the command line.

The tablet weighs less than 400g with a quad core 1.3 GHz processor with 1GB of RAM (less than the Samsung) with a reported (and tested) battery life of ~9 hours of active use (techradar.com, 2013) and can run low graphically intensive applications without affect in tropical rainforest conditions.

The Google Nexus tablet, the same as the Samsung Galaxy S4 smartphone, is chargeable from the micro USB power port with a standard 5v USB power supply, has the same reinforced glass, and very similar operational temperature and humidity ranges.

The power bank used is the mini Powergorilla with a 9000 mAh capacity (for comparison the Samsung and the Nexus have battery capacities of 2600 and 3950 mAh respectively) that can output voltage upto 19 volts, is compact, rugged and lightweight with a cost ~$40.

The amp and voltage compatible, lightweight and powerful solar panel used to charge the powerbank is the Anker PowerPort Lite 15W that cost ~$45 when purchased. The Anker solar panel is tested to charge in cloudy rainforest conditions, and through interruptions, and is incredibly durable (outdoorgearlab.com, 2017) and can charge two 5v USB devices at the same time (in the brightest conditions).

Collectively the hardware, peripherals, power bank and solar panel are durable, portable, lightweight, and powerful and with the right weather conditions, and operation, have enough power to never really have to go offline.

The total cost of the hardware (including the Samsung smartphone) totaled at around $180.

Chapter 4 Testing

Testing was conducted throughout and after the implementation of the 3 main components of the RIS but is summarised, with additions, and described more formally in this section and the 'Results' section.

To test the software, hardware and the prototype as a complete RIS different techniques were used :

- Usability testing — For its ease of use and operation.
- Unit testing — To check modules for bugs and internal states and working.
- Integration testing — That the parts of the system work together.
- System testing — That the system meets the requirements specification.
- Acceptance testing — Validation from the end user of the RIS.

To test and evaluate the prototype, each individual component was unit tested in isolation within their development cycle's testing phase and then tested to integrate with other components as they were completed to verify they were built correctly.

Additional testing and evaluation was conducted with lab based tests and field trials to evaluate the accuracy and technical performance of the RIS.

It was also possible to retrieve the serialised Track objects from the input component's locally stored files, query the database with SQL from the command line and use the complete Information Systems feedback loop from the output presentation component to perform and evaluate the results of the different testing activities.

With the above, the technical performance, accuracy, comparison with the project's requirements and the validation of the project's aims and objectives were able to be evaluated.

Technical and non-technical aspects of the system were black and white box evaluated from both the perspective of users and myself.

4.0 Input Component

A detailed description of the input components iterative testing phases can be found in the section '3.5.8.3 Testing and Iteration Retrospective' and '3.5.9 Iteration Two' but admittedly formal documentation was not kept to document the testing activities of the different testing phases for the first or second iteration of the input component but informal notes and importantly the results of the testing were recorded.

The formal conclusion of the input components unit, integration, usability and acceptance testing identified these points as needing to be addressed in the future iterations of the input component and the project:

- The Android application's 'toast' messages to inform the user that 'Files sent' is being shown for each file sent not just once, giving the appearance of the application being frozen (as identified in user testing).

- To better understand and improve the battery consumption of the Android application (as identified in unit testing).

- Additional parameters for the tracking session's Track and Point data objects could be user defined, or adjustable, to increase the usefulness of the RIS. Including UI changeable parameters for the interval between the recorded GPS coordinates and different selectable options for the accuracy of how they are generated (as identified in acceptance testing within interviews I attended).

- An implementation of the complete set of available smartphone hardware sensors, possibly with an optional UI selection component to select which sensors to record for a tracking sesion. This functionality is already partially implemented and was identified from user opinion to be the most desirable functionality to implement next (as identified in acceptance testing within interviews).

- An additional 'screen' of the Android application could be added to incorporate an online and offline implementation of the 3D globe of the output presentation component to display the current recorded tracking data collection session and all tracking sessions saved on the phone locally (as identified in system testing to increase the immediate feedback from the Information System feedback loop).

4.1 Central Processing and Data Storage Component

Additionally to the testing described in the sections '3.6.1.3 Server Unit and Integration Testing' and '3.6.2.7 Database Testing and Retrospective' some rudimentary JUnit5 testing was conducted on the individual methods and layers of the server but was not continued as it was identified that it was unlikely to increase the performance of the component or add value to the prototype when compared to the time taken to write the Junit5 tests.

JUnit5 tests were begun for the Track.java and Point.java classes. First testing individual getters and setters, and then the class's methods but it was quickly concluded that conducting more selective testing would make more economically and effective use of the time available by focusing on more key aspects of the system's main use case and the important key functions within the layers of the server.

JUnit5 was explored to test the Data Access Object layer but testing proved more complicated and time consuming than expected, the complexity was mostly due to understanding the @Autowiring annotation of SpringBoot rather than testing the parameters and outputs of the methods and the testing of the central component reverted to command line SQL, interactions from the input and output components to the API and simple outputs to the Standard.out.

4.2 Output Presentation Component

The output component provides the feedback loop of the complete IS structure. The feedback loop made it considerably easier to identify and evaluate the accuracy of that data that is collected by the system as well as its technical performance from an overview dashboard perspective.

A description of the testing phase of this component can be found in section '3.7.1 Analysis, Design and Retrospective Stages' with a summary of the results of the testing and the changes made to the component at the end of the '2.7.3 Web Page Summary' section.

With the extra time available before the projects submission date these three points were identified to be the highest priority to implement:

- The client side Javascript is clean but not as clean as it should be, including the clients approach to error handling and more modern JavaScript best practices.
- Extending the polylines method of the WebGlEarth API to draw polylines that do not need to connect two GPS locations. This is a main flaw of the current implementation of the component.
- A more focused presentation of data for how the system can more successfully meet the aims, objectives of the project and the requirements of system's users. Including mapping areas of rainforest not currently researched or under-researched, types of data needed to be collected, incorporating photos and audio, graphs, charts, infographics and contextualising the collected data with additional information about certain areas of the rainforest preserves and national forests present on the map.

The unit and integration testing of the output component did not indicate many bugs or runtime errors that could not be easily adjusted within the current iteration. As the final component implemented and tested, the results of its testing in many ways provided the results for the second iterations of all three components. The results of usability and acceptance testing for this component are described in more detail in the next section.

4.3 Usability Testing and User Interviews

Available users were asked to interact with the output presentation component webpage and the data collection input component application. The usability testing was somewhat hampered by the pandemic quarantine lockdown but I continued to pursue feedback from prospective job interviews for software engineering positions I attended and others.

The general response was a positive response, and it appeared that most users were impressed with the webpage and application in some way.

From a user perspective the 3D globe was well received but with anticipation of future additions and developments. From a usability perspective it's usability was considered to be good but different opinions expressed different adjustments they would prefer to see in the component, including larger sized UI components, larger text sizes and a way to organise and select different Tracks, perhaps by date or by location.

In some ways there is an acceptance and validation of the system and the project's aims and objectives for the reception of the input and output components by being invited to interview with different environment science organisations based on the system video walkthroughs posted on Youtube that I attached to my job applications.

User feedback received about the data collection application additionally influenced the starting point for the application’s second iteration. With the main feedback being focused on the improved presentation of the interface, including a different background image and different presentation of the buttons.

It was mentioned that it would have been preferred to have made use of one of the popular Android application development frameworks such as React or Kotlin but I think this may have been due to an expectation rather than a response to the usability of the application.

Finally, although I did not receive authorisation to include a direct quote from the organisation, one response to my prototype system was exclaimed to be “bloody good” by I presume quite an accomplished top hiring manager at an unnamed City of London firm.

4.4 Field Trials of Data Retrieval and Accuracy in Tropical Rainforest Conditions

Three field test experiments were formerly conducted, in addition to many informal field tests, to record and evaluate the consistency and accuracy of the data retrieval function of the system and to test that the sensors and the GPS hardware would collect accurate data and at the correct intervals. Only two of the three formal experiments are included in the report to focus on the most important results.

Additionally the field testing would also be able to evaluate the systems performance to collect data in tropical rainforest conditions. This point proved quite difficult and cycling to nearby Kew Gardens and field testing the data collecting component in the tropical glass houses proved the most available way to conduct this test.

The data captured by the input component was transposed to the below tables and the GPS coordinates recorded and plotted on Google maps to compare the system's accuracy with an external source before being networked and mapped on the output presentation component's 3D globe.

Sensor readings were recorded for each interval and compared with official temperature and humidity readings taken from Timeanddate.com and Wunderground.com weather report websites. The tracking session's intervals did not correspond exactly to the website's intervals and the closest comparative times were recorded and noted in the results. It was also observed that both websites appeared to filter anomalous readings from their websites so that there were no temperature or humidity readings available for certain times.

4.4.0 Field Test Experiment 1

Time intervals were set internally at 15 minutes using Android Studio. The application performed consistently capturing each Point 15 minutes apart. One anomaly observed was the date of 'Point 3' that is a different year, month and day but maintained the correct interval time.

Track	Date / Time	Temperature and Humidity	Timeanddate.com location: 'London'	Wunderground.com location: 'Hounslow' (West London)	Notes
Point 1	21.3.23 16:14:34	24.90 °C 56.49 %	13 °C (@ 16.20) 67 %	12 °C (@ 16.20) 67 %	Inside backpack
Point 2	21.03.23 16:29:34	38.23 °C 62.11 %	Not applicable	Not applicable	Tropical glasshouse
Point 3	01-10-07 16:59:33	20.90 °C 64.17 %	13 °C (@ 16.50) 63 %	12 °C (@ 16.50) 67 %	Anomalous date
Point 4	21-03-23 17:14:33	21.89 °C 60.45 %	12 °C (@ 17.20) 67 %	12 °C (@ 17.20) 67 %	

Table 5: Results of field experiment 1

The two comparison sources show a very slight variation in their temperature readings that are not recorded to decimal precision and a greater variation in the comparative humidity readings but are mostly consistent.
In comparison with the application's sensor readings the smartphone application recorded temperatures consistently ~11 degrees warmer.
'Point 2' was recorded inside the tropical rainforest glasshouse so is not compared with the two sources and did record the significant increase in temperature within the time tested inside the tropical rainforest glasshouse.

The humidity recorded by the application was relatively more accurate. The first reading was recorded with the smartphone inside mybackpack identifying a difference between backpack humidity and the humidity of the surrounding air. The following humidity reading in the tropical rainforest glasshouse could not be evaluated in comparison to the other sources but did not register much noticeable difference to the previous and next intervals. The third and fourth Points showed reasonable corresponding accuracy with the comparison sources.

The GPS location data was evaluated by manually plotting the Track's Points on GoogleMaps as illustrated. The blue markers indicate the Google Maps location of the GPS coordinates for Points 1, 2, 3 and 4. The red-lines indicate the distance and location to the actual locations of the Points recorded. These were noted on the Kew Gardens map I kept with me during the field tests.

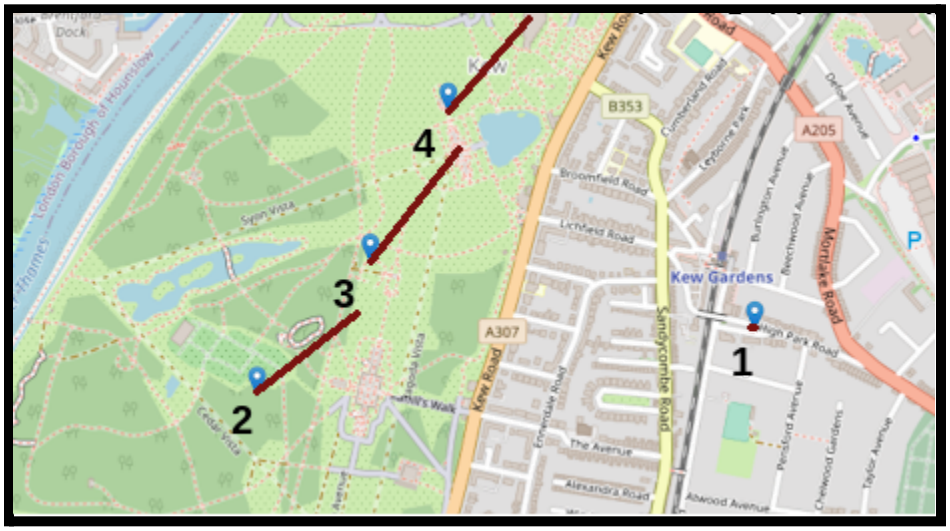


Fig. 24: GPS locations of field experiment 1

Point 1's accuracy appears to be very accurate with the way I cycled to Kew Gardens but I recollect initiating the tracking when I arrived at Kew Gardens and not before. I suspect this is because the application may retrieve the current GPS value stored in the operating system for its first value before requesting the current location by the second interval.
Point's 2, 3 and 4 appear to share a consistent inaccuracy of their direction and distance in their Latitude and Longitude of about 50 to 100 metres South West of the location I recorded on my map. This may be due to an inaccuracy of the devices hardware, their software stored values or an inaccuracy in the way that Kew Gardens is depicted in Google Maps.

Although the environmental conditions increased within the tropical rainforest glasshouse no noticeable effects of the tropical rainforest conditions on the smartphone hardware were observed.

4.4.1 Field Test Experiment 2

Following the findings of the first field test it was concluded that more intervals should be recorded and more time should be spent in the tropical rainforest glasshouses to more accurately evaluate the input component's performance. The initiation of the tracking should also start from a known condition and at the start of my cycle to Kew Gardens to increase the total number of intervals recorded and to observe the applications GPS accuracy over a greater distance and time.

It is worth noting at this point that these field tests were conducted in March and the first field test experiment temperature was ~10°C so in some ways limited the time I spent walking around in the cold in the first test.

Again the interval was set internally at 15 minutes. The interval times recorded in this test again proved very accurate, except an interval was not recorded at time 14:51:13 between Point's 4 and 5.
There was also more variation of the 'second's' value's accuracy, consistently recorded at 33 and 34 in the first test.

Again the date was consistent throughout but with a single anomalous value with an unknown correlation to the previous anomalous date or the expected date.

In this test the temperatures recorded by the system were considerably and consistently more accurate with the comparative sources.
The first temperature was recorded indoors and every temperature not recorded inside the tropical rainforest glasshouse being within ~1 degree of the official temperature recordings and in 1 case between the two sources.

The humidity values of the comparative sources show a variation ranging from a 5% to 1% difference but only 2 of the 7 intervals recorded were relevant for comparison. The other intervals were either recorded indoors in the tropical rainforest glasshouse or in my backpack.
Point's 3 and 4, the only two intervals that could be reasonably compared, showing between a 7% and 22% inaccuracy with the comparison sources data.

This second value conflicts with the earlier accuracy of the humidity from the first field test and could possibly be considered an anomaly or perhaps Kew Gardens being located by the river Thames and frequently gardened has differing humidity to the surrounding geographic area.

The Points recorded in the tropical rainforest glasshouse showed an increase in temperature and a drop in humidity that did not appear anomalous but without comparative readings from a calibrated device or an official reading source from Kew Gardens for the glasshouse the readings could not be evaluated.

Track	Date / Time	Temperature Humidity	Timeanddate.com Location 'London'	Wunderground.com location 'Hounslow' (West London)	notes
Point 1	01-08-13 13:49:56	23.83 °C 60.71 %	18 °C (@13.50) 60 %	17 °C (@ 13.50) 55 %	Date Anomaly Indoors
Point 2	21-03-29 14:05:04	19.85 °C 64.82 %	-	-	No comparative source data available.

Point 3	21-03-29 14:21:09	18.87 °C 63.35 %	19 °C (@14.21) 56 %	18 °C (@ 2.20) 52 %	
Point 4	21-03-29 14:36:13	17.29 °C 66.16 %	20 °C (@14.50) 49 %	19 °C (@ 2.50) 46 %	
Point 5	21-03-29 15:05:02	26.55 °C 49.59 %	Not applicable	Not applicable	Tropical Glasshouse
Point 6	21-03-29 15:22:20	27.78 °C 45.26 %	Not applicable	Not applicable	Tropical Glasshouse
Point 7	21-03-29 15:37:29	18.60 °C 68.02 %	20 °C (@15.50) 38 %	19 °C (@ 15.50) 37 %	Inside Backpack Outside Glasshouse

Table 6: Results of field experiment 2

The GPS coordinates were again plotted manually on GoogleMaps and compared to the Kew Gardens map locations I kept note of when the tests were conducted.
The Latitude and Longitude of Point 1 and Point 2 were recorded with exact accuracy from where the Track began and the route that I cycled to Kew Gardens (down to the cycle lane on the left hand side of the road I cycled on).

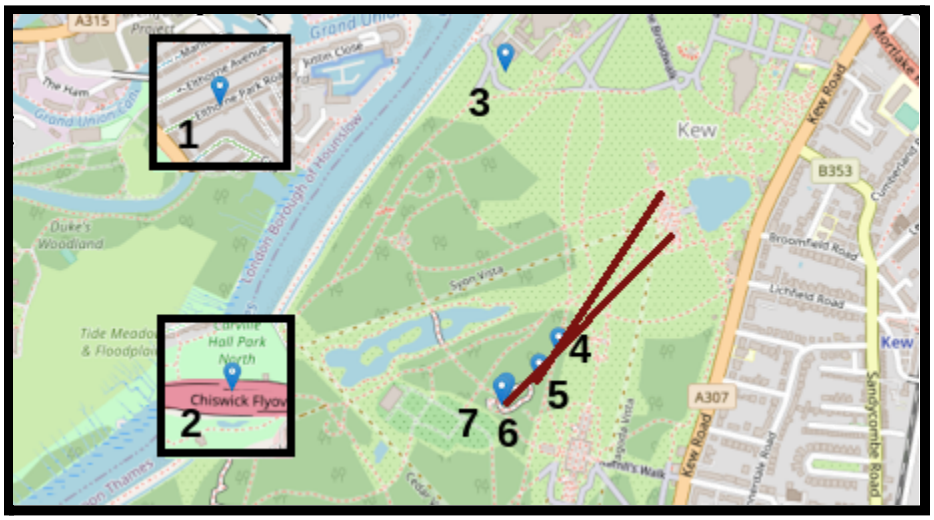


Fig. 25: GPS locations of field experiment 2

Points 5 and 6 are again indicated with red-lines to indicate their actual location within the tropical rainforest glasshouse and show a very similar distance and direction from their actual locations that was observed in the first field test.
It appears that the GPS locations recorded in test 1 and 2, that were recorded on a street, were exact, but locations that were recorded within Kew Gardens appear to share the same similar inaccuracy of about 50 - 100 metres in a South Westerly direction.
I suspect this may be due to the way Kew Gardens is depicted on Google Maps or possibly because the tropical rainforest glass houses are located on top of a hill ?

Points 3 and 4 were relatively very consistent with the locations I recorded on my map with only a variation of ~20 metres difference at the most.

Again there was no observable environmental effect of the tropical rainforest conditions on the hardware of the system. The temperature in the tropical rainforest glass houses was recorded around the average for a tropical rainforest of ~28°C as researched in the literary review with the humidity being noticeably lower than outside of the glasshouses and the average for a tropical rainforest.

4.5 Lab Testing

To further test and understand the accuracy of the temperature and humidity sensor readings, lab testing was conducted to compare the system's results with a Gellvann DC102 LCD digital temperature-humidity metre and clock (available on Amazon with a 4.4/ 5 star rating) to compare readings in a controlled environment over a greater time than the field experiments and in a stationary location. The GPS recorded the correct location throughout.

Two tests were conducted, one in a central heated room where the heating was set to increase to its greatest setting of 30°C, and the second by leaving the Gellvann DC102 and the input component outside on a windowsill over 3 hours.

The temperature in the room test started at 21°C, which showed on both the input component and the Gellvann DC102, and the room slowly heated to the maximum heat setting of the heating unit.

The Gellvann DC102 gave readings for the temperature lower than the input component and considerably colder than the perceived temperature in the room. The Gellvann DC102 was also observed to have slower to change temperature readings, that appeared to be internally set to a different or adaptive interval and did not register the changing conditions in the room as responsively as the input component.

No anomalous time or dates were observed in this test and the Gellvann DC102 and the input component recorded a measured response to the changing environment in the room. The humidity and temperature slowly increased which registered on both devices although the Gellvann DC102 showed a slower and apparent inaccurate measurement of the hot temperature in the room.

Centrally Heated Room Test

Time	Gellvann DC102	Input Component
1.45pm	21.4 44	21 46.81
2	21.4 44	22 46.53
2.15	22.5 44	23 46.99
2.30	22.5 44	24 46.75

2.45	22.5 44	23 46.63
3	23.2 44	24 46.16
3.15	23.2 44	24 46.61
3.30	23.6 44	25 46.31
3.45	23.6 44	25 46.72
4	23.4 44	26 47.81
4.15	23.4 44	26 47.24
4.30	23.9 44	26 47.01
4.45	23.9 44	26 47.65
5	23.9 44	26 46.51
5.15	24.4 45	27 46.53
5.30	24.4 45	27 47.37
5.45	24.4 45	27 47.55

Table 7: Results of lab test 1

Outside

The outside test showed a similar registering of the temperature and humidity on the wet and cold day with the Gellvann DC102 again indicating a slightly colder temperature and slightly lower humidity.
The input component at its highest humidity measurement displayed 100% humidity for 4 consecutive intervals and the test was concluded with this important observation.

Time	Gellvann DC102	Input Component

6pm	11.5 87	13 89.46
6.15	11.5 87	13 98.23
6.30	10.6 86	13 100
6.45	10.6 86	12 100
7	10.1 89	11 100
7.15	10.1 89	11 100

Table 8: Results of lab test 2

From the two tests it appears that the accuracy of the input component's temperature and humidity sensor readings have a comparable accuracy to the commercially available Gellvann DC102 digital temperature-humidity metre and clock. With the input component observed to be more responsive to a rapidly changing environment. This made me reconsider the results of the field tests as the comparison sources show humidity and temperature measurements that were not observed to change or to only change very gradually.

The results of the testing are explored in the next section.

Chapter 5 Results of Use

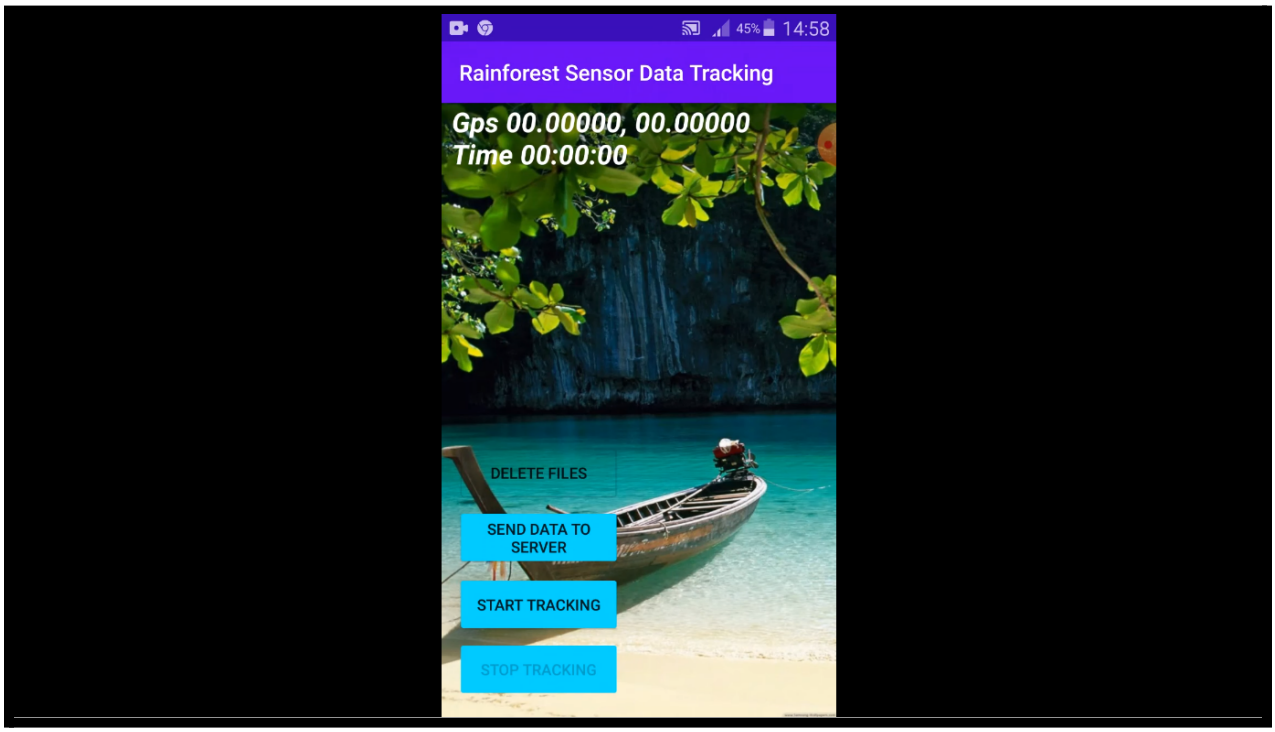
The use-cases defined in the initial stages of the project and later refined and used in the more detailed designs of the components can be used to evaluate and validate the resulting complete prototype.
The prototype should be able to perform the refined use-cases validating it can perform the identified users needs.

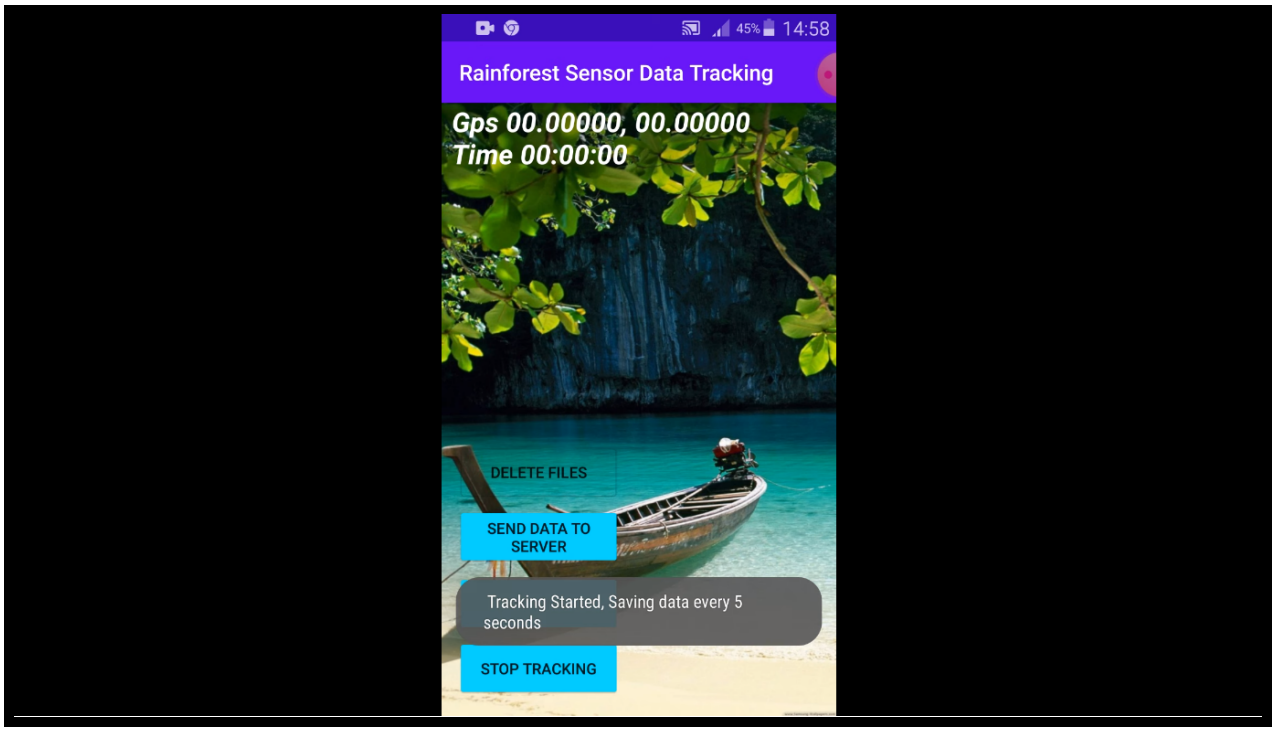
5.0 Input Component Video Walkthrough

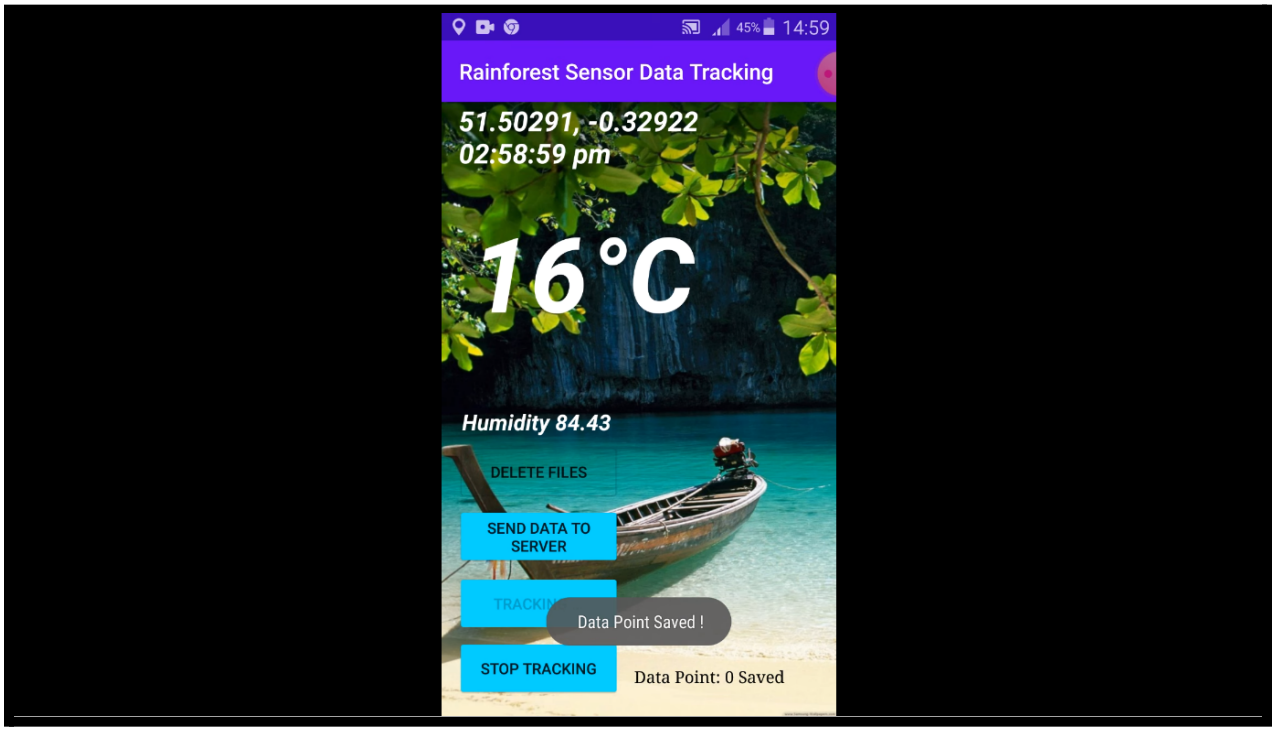
https://www.youtube.com/watch?v=Cb32cDj3JFc&ab_channel=markstart00

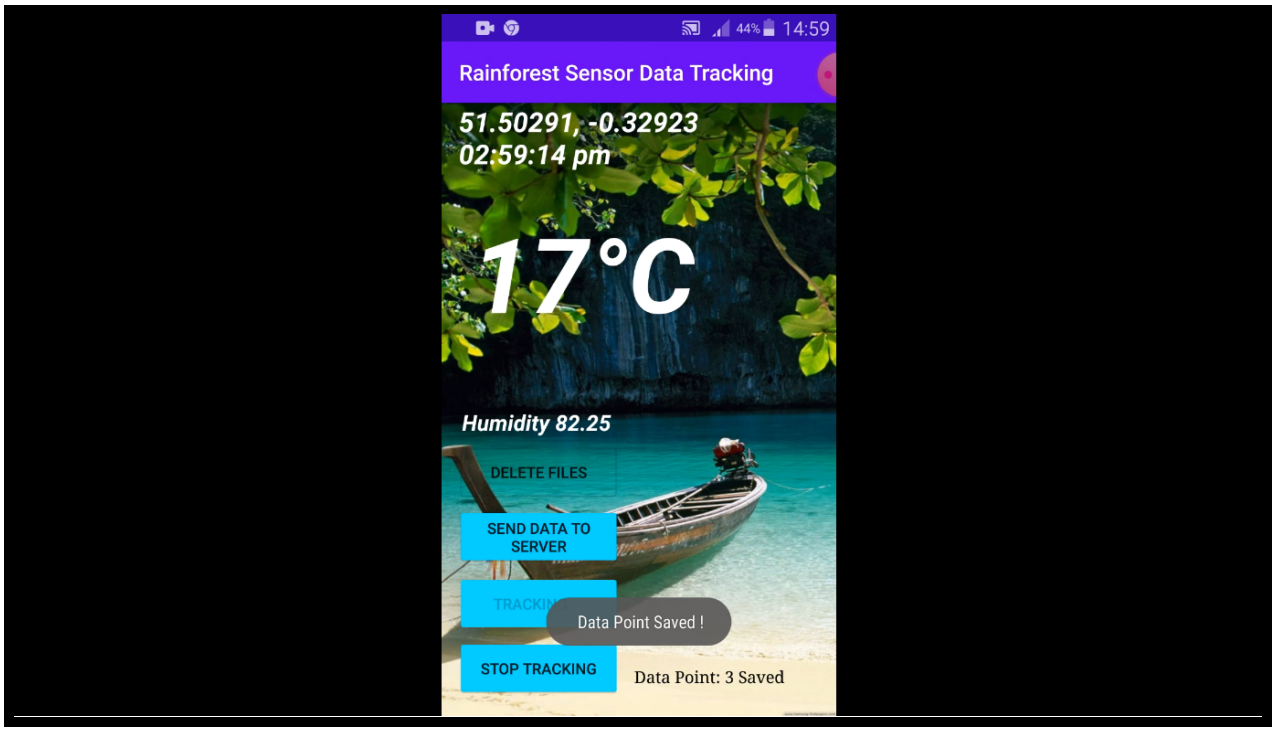
This first video walkthrough exhibits the complete implementation of the 'Record Track Point Sensor Data Use Case'.
The user can initiate tracking, stop tracking (with data being automatically stored locally), send the data to the server and delete the locally stored data.
The user is informed of the internal actions of the application and receives feedback from user interactions following some simple interaction design feedback principles.
The presentation of the application and data is clear and well presented. The precision of the GPS coordinates, temperature and humidity values shown is shorter than the degree of precision stored internally this keeps the UI simple and clear.
The application is robust, fast and performs without errors for all operations of its functionality.
From personal and user feedback the Input Component received positive feedback for both its presentation and usability.

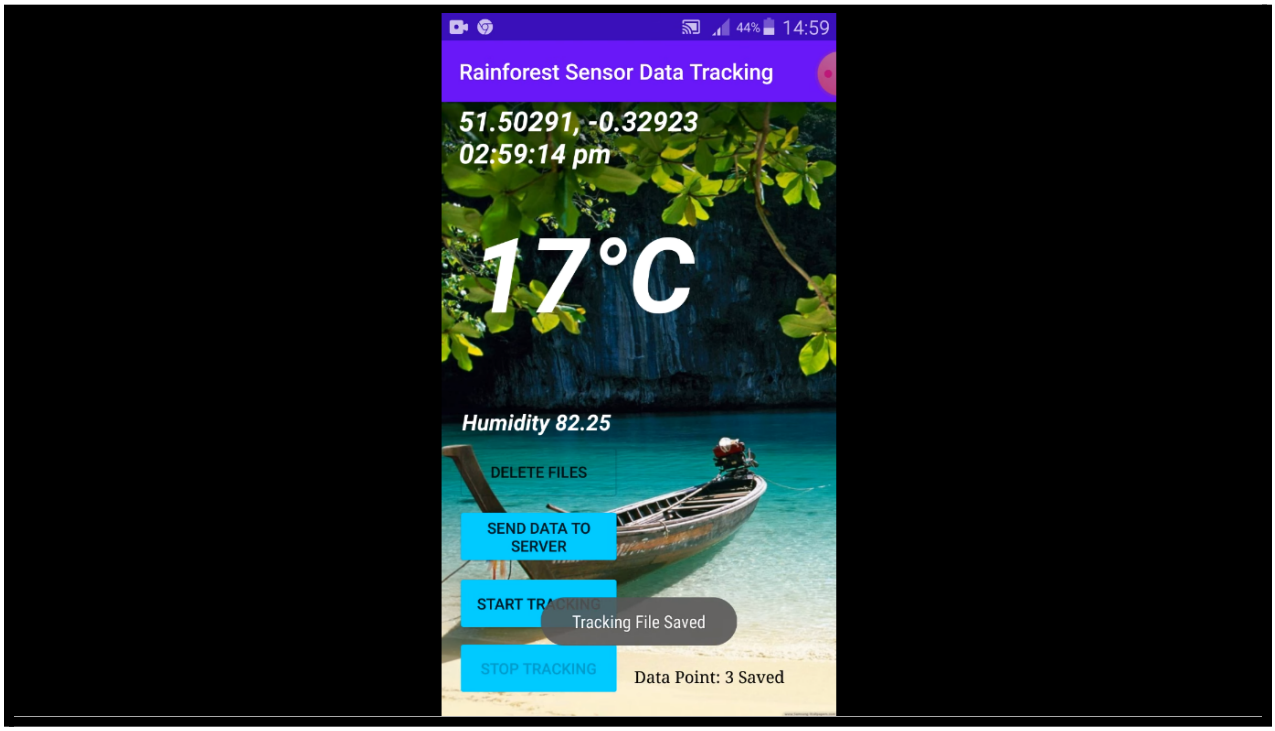
Sequential screen captures of the walkthrough are also included below.
* For the walkthrough the interval between Points is set to 3 seconds for demonstration purposes.

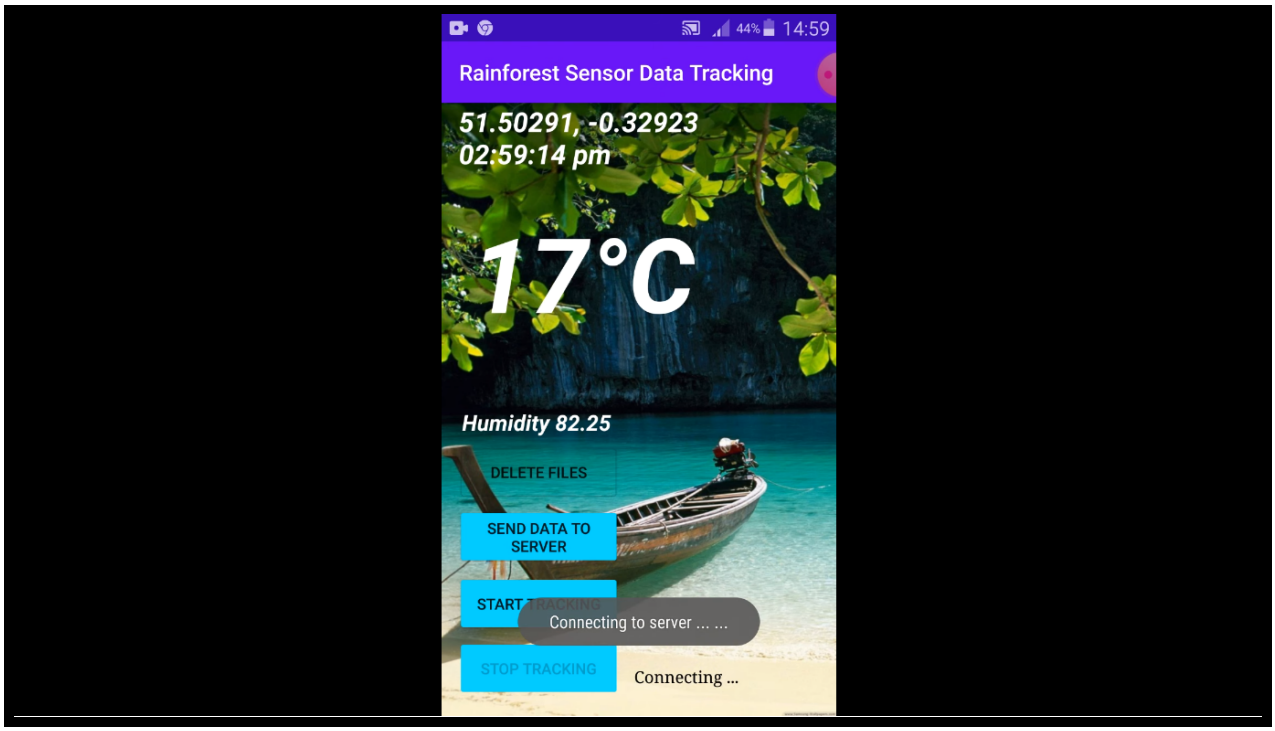


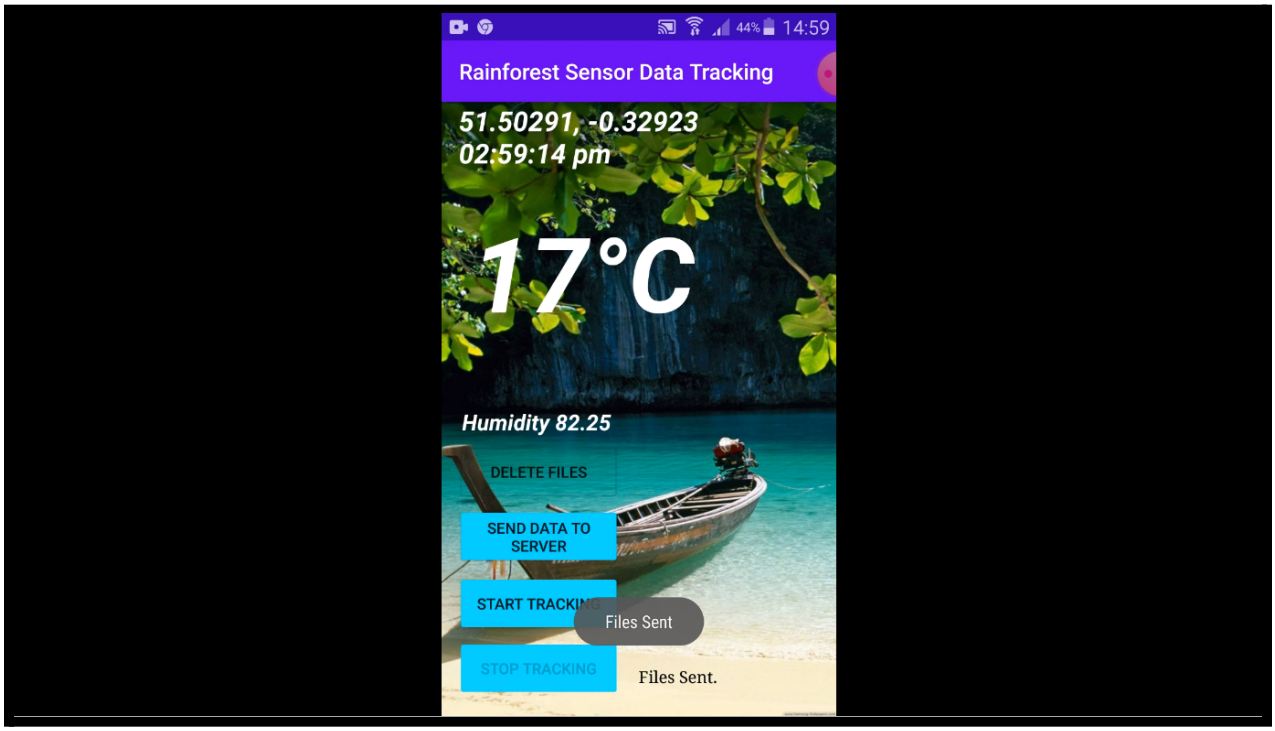

45% 14:58
Rainforest Sensor Data Tracking
Gps 00.00000, 00.00000
Time 00:00:00
DELETE FILES
SEND DATA TO SERVER
Tracking Started, Saving data every 5 seconds
STOP TRACKING

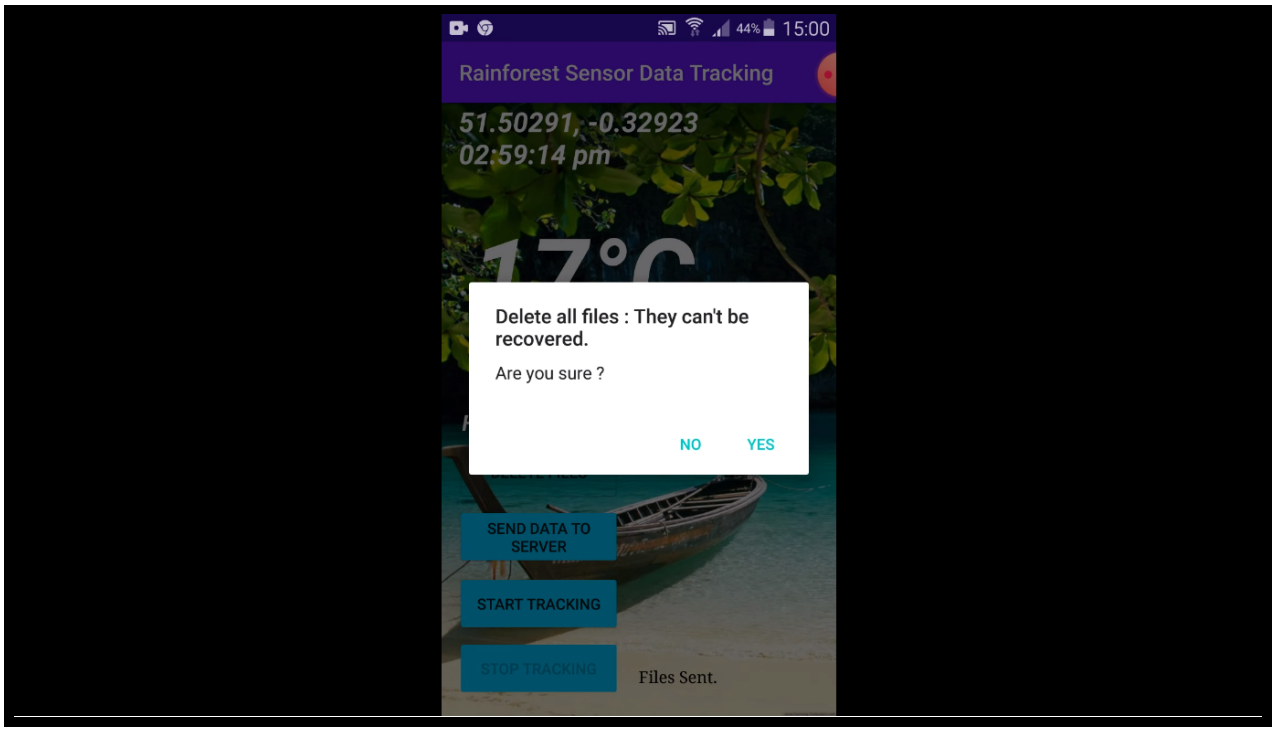

45% 14:59
Rainforest Sensor Data Tracking
51.50291, -0.32922
02:58:59 pm
16°C
Humidity 84.43
DELETE FILES
SEND DATA TO SERVER
Data Point Saved !
STOP TRACKING
Data Point: 0 Saved


14:59
Rainforest Sensor Data Tracking
51.50291, -0.32923
02:59:14 pm
17°C
Humidity 82.25
DELETE FILES
SEND DATA TO SERVER
Data Point Saved !
STOP TRACKING
Data Point: 3 Saved


14:59
Rainforest Sensor Data Tracking
51.50291, -0.32923
02:59:14 pm
17°C
Humidity 82.25
DELETE FILES
SEND DATA TO SERVER
Tracking File Saved
STOP TRACKING
Data Point: 3 Saved


14:59
Rainforest Sensor Data Tracking
51.50291, -0.32923
02:59:14 pm
17°C
Humidity 82.25
DELETE FILES
SEND DATA TO SERVER
START TRACKING
Connecting to server ... ...
STOP TRACKING
Connecting ...


14:59
Rainforest Sensor Data Tracking
51.50291, -0.32923
02:59:14 pm
17°C
Humidity 82.25
DELETE FILES
SEND DATA TO SERVER
START TRACKING
Files Sent
STOP TRACKING
Files Sent.



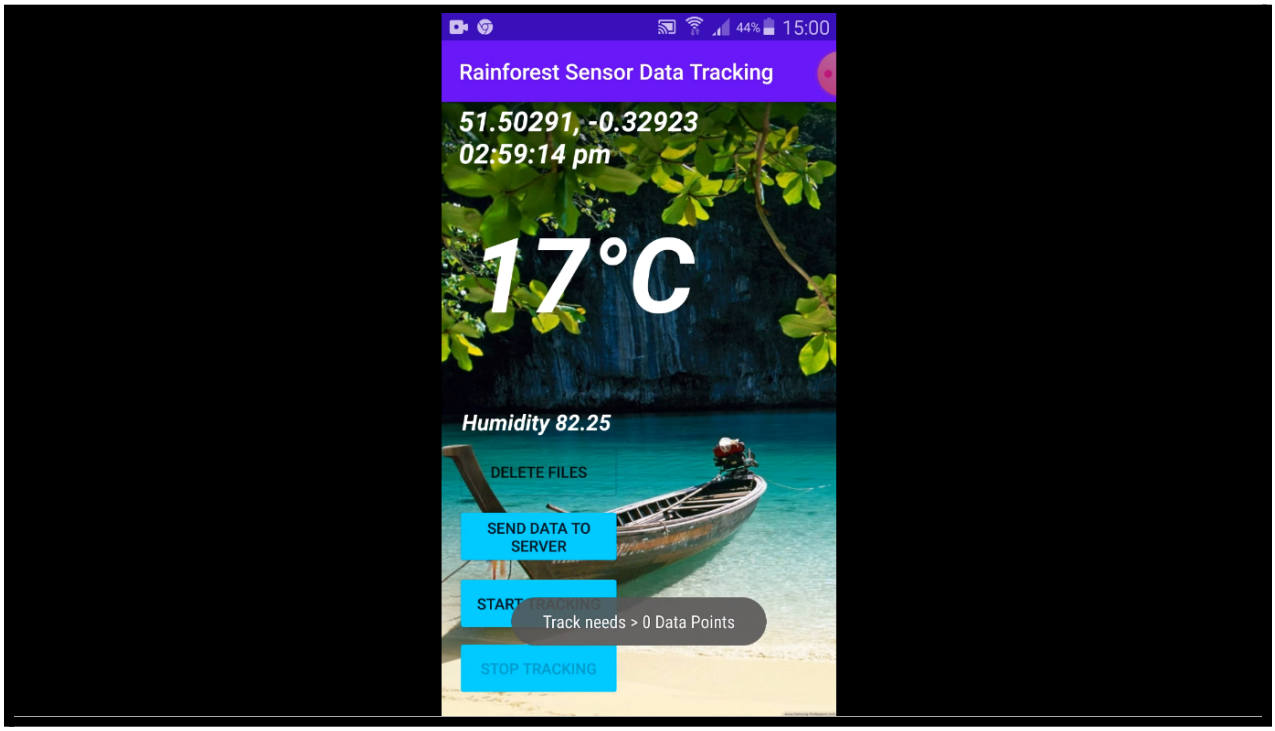


Fig. 26-34: Input Component Video Walkthrough

5.1 Processing and Data Storage Component Video Walkthrough

https://www.youtube.com/watch?v=kujqOj48kVg&t=472s&ab_channel=markstart00

This walkthrough shows the complete integration of both the Input Component, the Processing and Data Storage Component and the Output Component of the system.

Importantly at 6:35 of the video all Track's in the database are deleted, shown to not be viewable on the web page and the local Track and Point data stored on an offscreen Input Component are re-sent to the server and database. At 6:53 the server outputs to the command line the SQL of the newly inserted Point sensor readings into the point_sensor_readings table for the newly sent Track and Point data sent from the offscreen Input Component.

The walkthrough also shows the implementation of the 'Data Administrator' use case where a user with command line access to the database can create, read, update and delete data by executing SQL queries and commands. The use case was not detailed in the design or implementation sections of the database but is shown in the use-case diagram of the complete system and shown in the video walkthrough to illustrate the comprehensiveness of the prototype.

The walkthrough captures the running of the Postgres database Docker container together with the SpringBoot web server executed on the local machine as a .jar from the local Linux machine with the command line as side by side terminals.

The Docker Image is downloaded and the container (containing a base image of Linux and the Postgres database) is built, run and interactively executed, and the Postgres database logged into.
The Rainforest Database is created, the server started and the test data is interacted with SQL from the command line.
The walkthrough video shows the SQL commands that can retrieve joined Track and Point data (with their respective UUID's) from connected tables within the database as well as showing the actual degree of precision of GPS latitude, longitude, temperature and humidity that are stored within the system.

The video walkthrough shows how the same data (shown side by side) from the server and database are accessible from the Output Component 3D globe web page and that the corresponding command line SQL retrieved data is the same set of data that is presented on the 3D globe and that when data is deleted from the database it is no longer available on the webpage.

The unused test data that is not currently implemented to be displayable on the webpage is shown to be retrievable from the command line including Koppen climate information and continental regions and subregions.

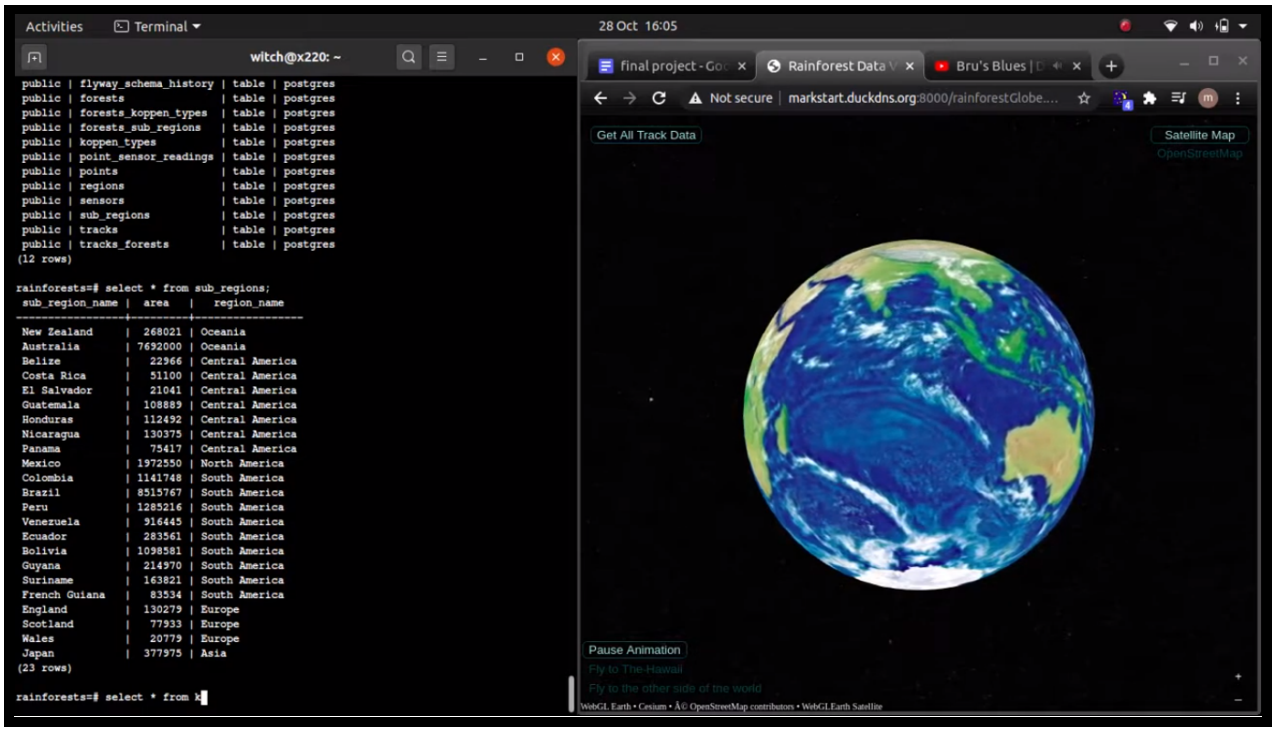


Fig. 35: Central processing and data storage component video walkthrough image 1

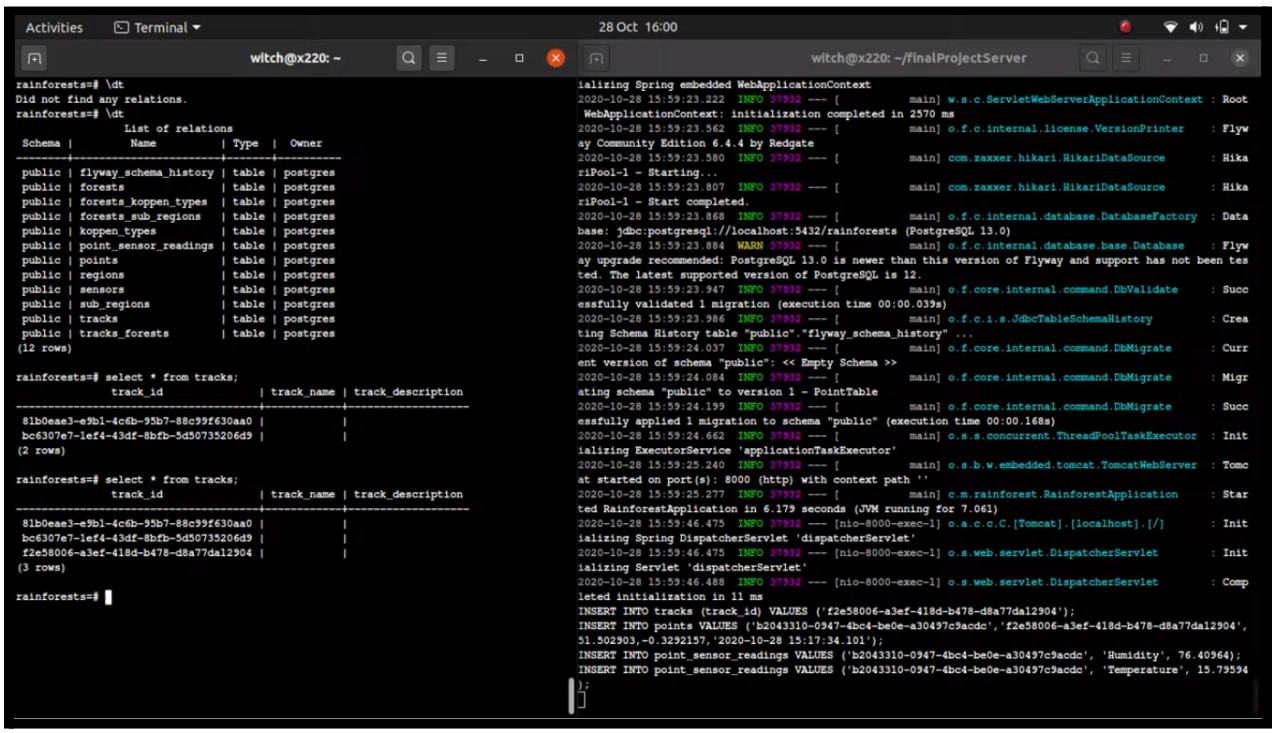


Fig. 36: Central processing and data storage component video walkthrough image 2

5.2 Output Component 3D Globe Webpage Video Walkthrough

https://www.youtube.com/watch?v=0TvVNaYX_e0&ab_channel=markstart00

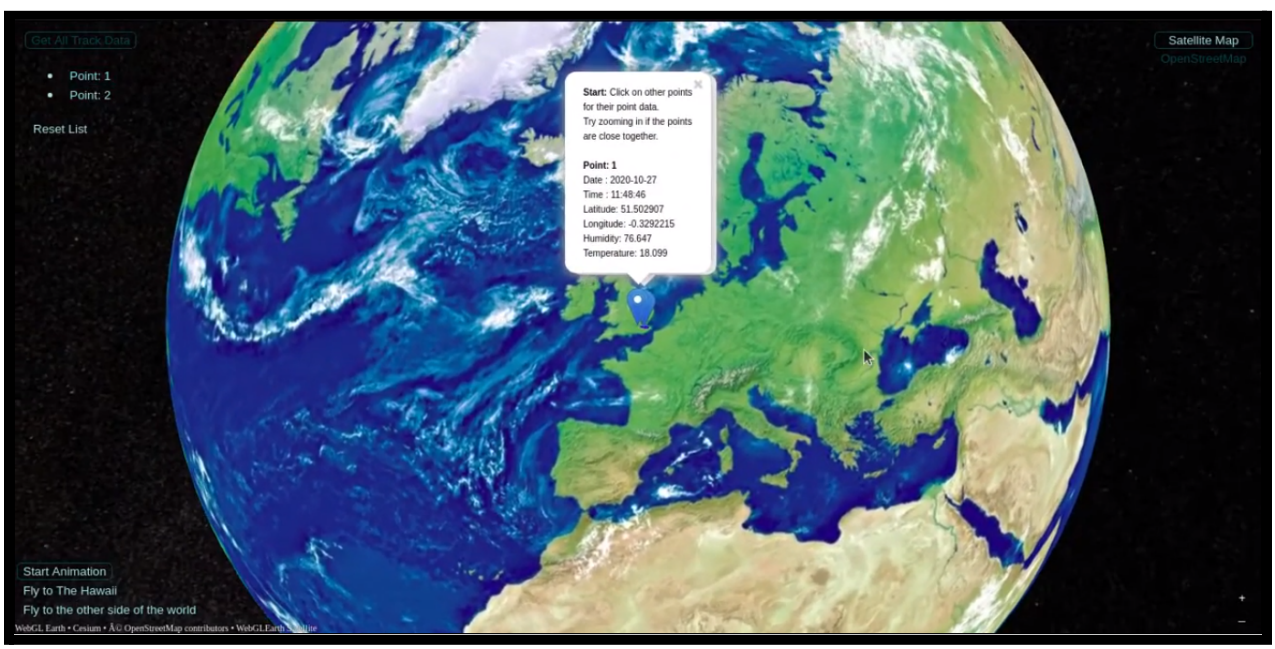
The final video walkthrough shows the webpage dashboard in more focus and the interactive functionality of the methods described in the output presentation component's implementation section.
The walkthrough shows the complete implementation of the 'View Track Point Sensor Data' use case.
The user can load the globe, select Track and Points to display, can zoom in and out, select different map overlays and can reset the currently displayed Track and Point information.
The use case extension that a user can select a subset of the data (an individual Point) is implemented by popup markers being open and closable and that the zoom levels can focus on a single Point of data.
There is limited control of the data's customization but the prototype webpage shows the breadth and depth of the prototype's functionality clearly and without error.

The webpage is the source of information for the feedback cycle of the Information System. The current implementation of the visualizable data is considered adequate and reliable for further feedback to improve the performance of the system and the presentation of its data.

The map tiles and zoom levels provide a comparative level of detail to GoogleMaps including the location of biological rainforest preserves and national forests to a significant detail.

The rotation and pause animation function works seamlessly as do the select Track, Point and reset data menu interactivity.

The transparent presentation theme of the buttons increase the field of view of the globe and the Point data and contrast nicely with the Milky Way background.



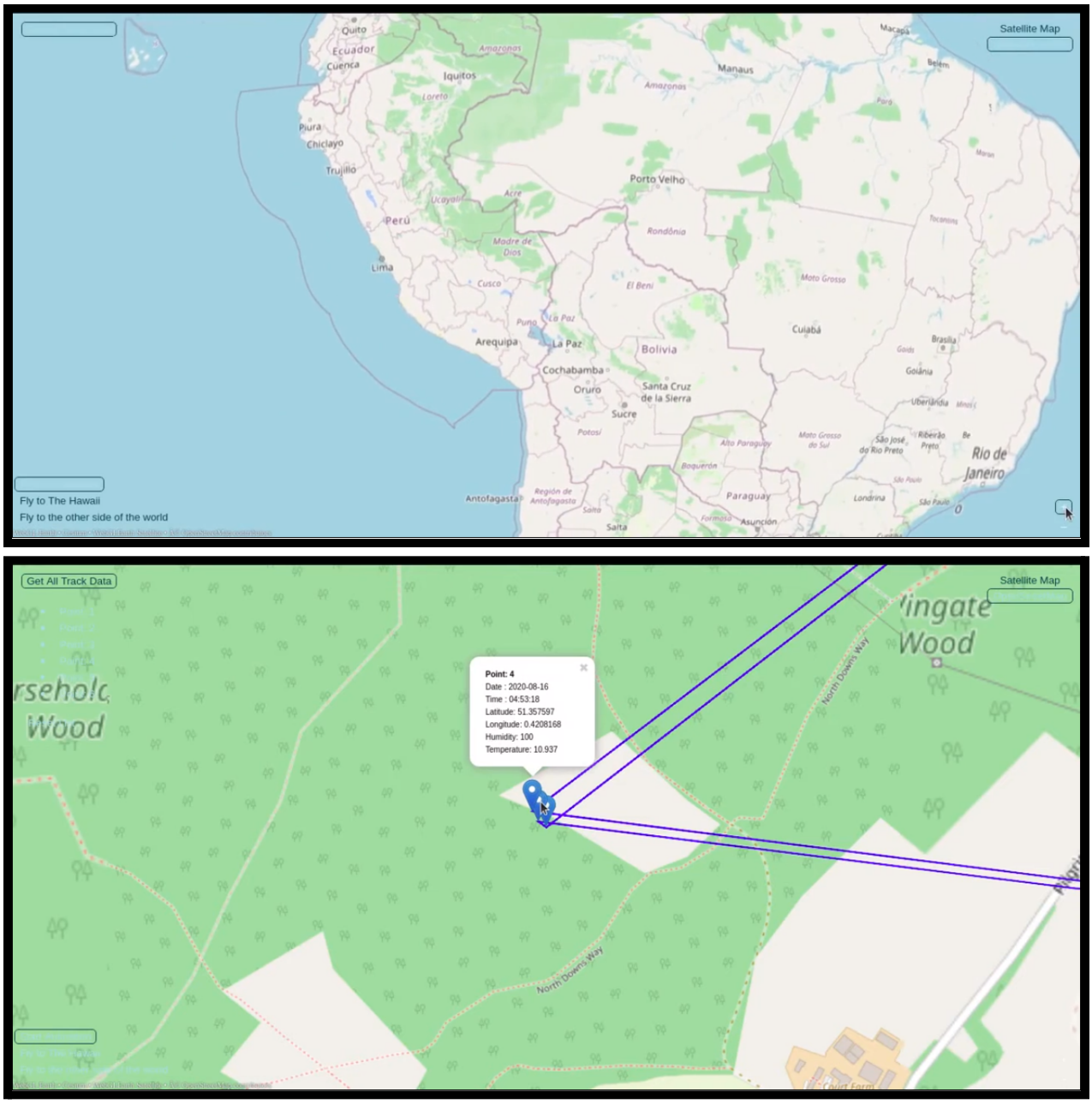


Fig. 37-39 Output presentation component video walkthrough

5.3 Results of System Testing

A formal system testing activity was completed to evaluate that the implemented prototype system objectively parallels the requirements of the requirements specification. The requirements are used to validate the current implementation of the prototype.

In comparison with the requirements of section 3.6 it was considered that a very high percentage of the software, hardware, usability, functional and non-functional primary requirements were achieved as well as some of the secondary requirements in the prototype. The following paragraphs correlate to the requirements in the requirements section.

The main project specific requirement to develop a small scale prototype of limited scope displaying an implementation of a set of vertical and horizontal functionality was completed within the time frame. This was important so that the testing and evaluation of the prototype could reveal early strengths and weaknesses.

The system specific requirement to automate the data collection for a variety of sensors at the same, or different locations, and to make possible the networking of the collected data to another computer in, or not in the field, was implemented well with the client-server architecture making the data transmittable from any location from the field where an Internet connection is available.

The 3D globe web page provides an interactive web interface to visualise and select a set of the system's data and importantly completed the IS feedback loop that can be used to evaluate the accuracy and performance of the Input Component, particularly the GPS, temperature and humidity sensor's accuracy.

The contextual information programmed into the database that was not made use of and could be a viable way to retrieve accurate comparative data from reliable sources that could be used to compare and evaluate the system and its components accuracy.

Successful reuse of recycled hardware and the use of free and open source software and programming languages was achieved and showed no considerable negative effect on the performance of the system in comparison to the money saved. The total cost of the system came to about $180 or about £140.

The project's software is well documented in this report as well as being available on its own Github project page which greatly increases its accessibility for evaluation, review and feedback and even use!

A battery-performance balance analysis was not formally conducted but an exploration of battery consuming precision settings of the hardware sensors was investigated in the input components unit testing phase with a semi-detailed starting point for its implementation in a future iteration.

The prototype successfully implements a complete Information System structure with the 4 main key components including an input component, a central processing and persistent centralised database storage component, an output presentation component and a feedback cycle.

Each component was attempted to be implemented in a flexible, modular, maintainable and principled way that was as simply as possible following the advice identified from the research of the literary review sources.

All of the primary and secondary hardware requirements were found to be present in the hardware selected but the additional requirement to implement additional sensors such as a camera and microphone

were not attempted. The project structure and software however were designed in a way that it is easily extendable for the implementation of the built in sensors of the smart device.

The primary requirements of the software that were 'not' met were the requirements of manual data entry and the implementation of a terrain map tile layer, with all other primary requirements being achieved.

Of the secondary software requirements minimal data validation and filtering, including anomaly identification were implemented with the exception that a Track of 0 Point's can not be stored and non-unique IDs cannot be entered into the database.
Reports are not autogeneratable and functionality to customise the visualisation of the data is limited by the output components current implementation.

The requirement of usability without literacy was not addressed.

The data model was well thought out, designed and implemented and aligns well to the scientific purpose of the system. A key point to make is that the Sensor_Readings table in the database is implemented in a way that additional sensors can easily be added to the system. This planned extendability is a key success of the system although can only be tested or proven to work with the implementation of additional sensors.

Maybe one of the least tested areas of the system is the ability to operate in tropical rainforest conditions but as a prototype, the system was tested to work briefly in artificial tropical rainforest conditions in the tropical rainforest glasshouse in Kew Gardens and registered the increase in environmental conditions.

The Nexus 7 in-the-field Linux laptop, having been tested in tropical rainforest conditions prior to the project, and the operational temperature and humidity ranges of the Samsung smartphone, which is designed to work in a range of geographical locations, in some ways limits the impact of the system not being comprehensively tested in this area.

The achieved additional non-functional requirement to make the system easy to use and to automate as much of the system as possible appeared to disappoint some of the users that reviewed or tested the system with some users, when evaluating the complexity, would have prefered a more sophisticated user interface or a more impressive complicated system design.

Finally, although surprisingly, no runtime errors were found in the testing of the system, error messages were implemented to be clear and informative when or if encountered.

The system testing indicates that the prototype design and implementation achieves a very high percentage, but not all, of the primary and some of the secondary requirements from the requirements specification.

Chapter 6 Discussion of Results

6.0 Discussion of Test Results

The amount of testing completed was comparative to the time spent on the research, design and implementation phases of the prototype with the intention that the faster the development cycle was completed the faster user feedback, evaluation, system testing and the IS feedback cycle could provide the input, starting point and the requirements for future improvements and iterations.

Testing was focused on key aspects of the system. Unit testing was conducted so that components would operate accurately and without error with the results of the testing not revealing any major flaws in the design or implementation of the individual components.
Integration testing and the system-wide networking of the components into a complete system was completed so that the workflow and user tasks of the system could also be performed accurately and without error.

Some simple test cases were possible to be derived from the use cases of the system and the requirements of the individual components and the project, and explored in the video walkthroughs of the components to illustrate the complete working functionality of the prototype RIS. Test cases were not recorded formally.

The results of the field testing experiments revealed some important baseline strengths and weaknesses of the Input Component.

It could be said that the data collected is only as good as the quality of the smartphone sensor's collecting the data but with only limited testing it is undetermined where the inaccuracies of the GPS, temperature and humidity sensors originates from. Further testing with a calibrated device or similar comparative source would be an ideal way to perform additional testing.

From the literary review of the ARGOS system and the comparative online weather sources used to evaluate the temperature and humidity readings in the field experiments it was shown that inbuilt filtering of anomalous data is incorporated into even the most sophisticated systems. And these issues in the prototype could possibly be resolved in the same way.

The lab testing results conducted in a stationary location and in a controlled setting highlighted that at high humidities the system registered 100% humidity consistently (perhaps because the humidity sensor became wet) and that the system's temperature and humidity readings are comparable to the commercially available Gellvann DC102 LCD digital temperature-humidity metre and clock.

The controlled environment of the central heated room further indicated a strength of the system that it is more responsive to temperature changes in a rapidly changing environment but still showed slightly different measurements to the comparison source used.

Overall from the limited testing results, the accuracy of the system indicated that the system has a GPS accuracy ranging from exact to within 50 - 100 metres in a South Westerly direction. Occasional anomalous time and data values. Can perform in artificial tropical rainforest conditions with temperature and humidity sensors reading accuracy comparable to a commercially available temperature-humidity clock but comparatively less accurate than the established online comparative sources used as baselines in the field trials.

Importantly the testing showed the low cost recycled hardware prototype system can be used as an in-the-field data collection system with a not completely understood accuracy.

6.1 Discussion of Usability, User Acceptance and User Interviews

Usability and acceptance testing were conducted (although limited with only informal feedback obtained) to evaluate the level of the quality of the implementation that the more objective functional and system testing could not easily evaluate.

In some ways the acceptance of the prototype could be measured by the invitation to interview at the organisations I applied to based on my CV and attaching the video walkthroughs of the project prepared for this report.

The general reception of the quality of the early stage prototype was considered high but subjective. Ideally a survey questionnaire of user interviews could be conducted but was limited due to the pandemic quarantine and other time commitments. Although less formal interviews with accomplished professionals did provide a level of qualitative insight that other quantitative approaches may not have.

Feedback from informal user interviews for the usability of the input component became the initiating source for the analysis and design stages of the second iteration of the input component which significantly improved its appearance, presentation and improved its performance in terms of user feedback, communication of the internal workings of the application, the operation and the logical progression of a user's interactions.

It was found to be very useful to have opinions and feedback from many different professional sources and the compromises made for it to be possible to conduct the limited usability, acceptance and user testing should definitely be improved in future projects or future iterations.

6.2 Discussion of Software and Project Methodology

The software backbone of the system performs well with no runtime errors observed and with only minor user interface bugs and improvements to code implementation to understand and address.

The software was divided into three components following the plan to implement the familiar IS architecture and careful analysis of appropriate and available technologies and accurate, principled and detailed designs were drawn before the implementation phases of the software. I expect these are the main reasons the software works to the level that it does.

Taking time to select, adapt and extend a suitable project methodology and to include useful methods and approaches identified in the literary review and from several other sources including the software engineering modules from the degree program and the recommended reading, significantly improved the structure and processes in the software's code. Including :

- Making early architecture decisions
- Using A modular approach to ease the control of complexity
- Meeting dates over requirements
- Stabilising requirements
- Diagrams too complex to hand draw are too complex

- The porcedual approach to designing and implementing the relational database
- An Object Oriented style of a lot of little objects with rich functionality.
- And to make use of programming Interfaces.

To avoid requirements changing and causing stages to overrun or possibly derailing the structure of the project's development, significant focus and time was spent on the literary review to interpret more accurate requirements that were then frozen.

Using both the iterative and waterfall process helped to keep the project on track, helped to focus its scope and prevented delays, and helped me to go back and fix my mistakes more quickly!

Formal testing phases and different types of testing significantly helped to improve and understand the software program code. I would have liked to have recorded the testing in a more formal presentation and to have completed more iterations of all 3 components.

I think the 3 main areas I would like to personally focus on are the JDBC server-database interactivity layer, implementing more sensors together with their UI interaction and more complex data visualisation and presentation in the output presentation component of the systems data, as well as collecting a lot more data!

6.3 The Cost and Accessibility of the Prototype System

Two key aims of the project are to address the cost and accessibility of Rainforest Information Systems.

With the completed analysis, design, implementation, testing and evaluation phases of the project it is possible to compare the prototype with the RIS and FIS systems and the other technologies that are used to monitor and collect field data in tropical rainforests that were researched in the literary review.

System	**Type**	**Automated data collection ?**	**Interference by vegetation**	**Cost**	**Accessibility**
Traditional Radio Telemetry	Tracking	No	Low	Medium	High
GPS / GSM Tracking	Tracking	Yes	Medium	High	High
Satellite Tracking (ARGOS)	Tracking	Yes	High	Very High	Medium
Automated Radio Telemetry (ARTS)	Tracking	Yes	Low	Very High	Very Low
Rainforest Connection	Monitoring	Yes	Low	High	Low
SMART Collars	Research	Yes	Low	N/A	N/A
Global Forest Watch	Forestry	No	Medium	Free	High
Prototype RIS	**Data collection**	**Yes**	**Very Low**	**Very Low**	**High**

Table 9: Generalised comparison of systems

The total cost of the system's hardware totals at about $180 (or about £140) and the only necessary hardware, the smartphone, is currently available on eBay for $40 (£30) with free shipping.

The cost of the input component (and complete system) is well below the researched cost of the ARGOS, VHF, GPS, GSM and other systems researched and towards the lowest cost of the recycled devices compared, with only the recycled fitness tracker researched to be cheaper.

The cost of the input component could be lowered even further by adapting the field data collection application to be installable on a range of different smartphones so new hardware is not needed to be purchased if a user already owns a compatible smartphone lowering its cost to free, the same as the Global Forest Watch application.

The issue with this approach is only a small number of cell phones (~25) (phonegg, 2021) were researched to have the required humidity sensor, seen as a necessary sensor to monitor tropical environments with naturally substantial amounts of very high precipitation and humidity.

The choice of hardware for the input component was made as this approach may be able to improve rainforest data collection as it is cheap and accessible as one of the top 20 selling smartphones of all time (Telegraph, 2017) and a smartphone with one of the most inbuilt sensors (gsmarena, 2013) increasing its usefulness at automating the collection of tropical rainforest data and it's availability and accessibility in developing countries, where most of the world's tropical rainforest are located.

The same recycled smartphone can also be used to access the output component presentation 3D globe with the packaged browser application and with the client server architecture only one central computer is needed for many input and output components.

The project's code is hosted on its Github project page as open source increasing its availability for use but currently requires technical understanding to deploy the application using Android studio on to the input component hardware.

The accessibility to the prototype RIS is considered to be greater than the researched academic ARTS system, the commercially available ARGOS system, the project specific Rainforest Connection system and the research SMART collar project that necessitates conderiable infrastructure, high initial and running costs, or are commercially unavailable, project specific or their source code not available.

Cloning the Github project page and purchasing the recycled input component hardware and the only barriers of entry to the system. The central component can run on any computer with Java and Docker installed and a 8000 port forwarded.

As much of the system is automated as possible, except possibly attempting to network the data automatically, and as such its ease of use is considered to be very high.

Unlike other systems the application is only available in English limiting its international use. The project formalised a secondary requirement of illiteracy for system use but this requirement was not explored in the current implementation.

Overall the system in comparison appears to fill a niche when compared to more expensive, sophisticated and academic systems.

Chapter 7 Conclusions

7.0 Current State of the Project

The project set out to understand how technologies, software, hardware and system architecture have to be specifically designed for an Information System to be applied and used in tropical rainforest conditions. More specifically how a Rainforest Information System retrieves tropical rainforest data and to investigate the socio-economic viability of using free open source tools and recycled hardware to reduce the cost and the barrier of entry of sophisticated, academic and expensive systems that operate in the tropical rainforests of the world.

There was an additional scientific aspect of the project to understand why the rainforest remains unexplored and under researched and how the application of the outlined requirements of this project's prototype system could automate the data collection and research of the world's tropical rainforest.

By completing a relatively comprehensive literature review within the time available of technologies used in tropical rainforest research including forestry, wildlife tracking, academic and scientific projects and systems with supporting research to compare and understand the market of free software applications and smart devices that can be used to collect field data (including an internship with a Rainforest Information System environmental sciences company based in San Francisco) the project's requirements and design for a prototype system that could meet the aims of the project were well researched.

The requirements of the project outline a system that goes beyond the current implementation of the prototype. The project methodology was chosen to complete a prototype of limited scope within the project's available timeframe but to be open to future iterations that could broaden the breadth of the prototype into a complete system. Currently only a small set of the system's possible sensors are implemented. This was intended so that a rapidly developed prototype could be tested and evaluated by different stakeholders of the prototype system to identify its early strengths and weaknesses.

The evaluation of the system was partly limited by the project being self-study and mostly conducted during the covid-19 quarantine. As a compromise Q/A sessions were informally conducted within job interviews I attended with Cambridge Environmental Research Consultancy, The Natural History Museum, The United Nations Environmental Conservation Centre and meetings I attended with Rainforest Connection.

The user reception and evaluation received was generally very positive of the initial prototype and provided very strong and informative feedback to be used as the input for future iterations and improvements to be made to the system. Closely working with potential system users and stakeholders was the most incomplete part of the project and something I would like to work towards in the future.

The design of the system and implementation were conducted in a very principled way and followed closely the advice researched from literary review sources that's aim was to provide advice for the successful completion of tropical rainforest Information System projects.

The design of the system was based around two key use cases to automate the collection of field data, network the data to a central server and make possible the visualisation and analysis of the data in an interactive web-based dashboard completing the Information System component structure and IS feedback loop.

Field experiments and lab testing explored the viability of the prototype system's recycled hardware and its operation in artificial tropical rainforest conditions and in comparison with the accuracy of two established online weather services and a well reviewed Amazon commercial temperature humidity clock.

The testing indicated GPS precision of the prototype ranging from exact to within 50 - 100 metres of actual location of the device. Other anomalous date, time and humidity values were observed in the field experiments but not in the lab based testing. These would be necessary to address in the next iteration of the prototype.

Importantly the testing indicated the prototype to be quite capable of operating in the field and within artificial tropical rainforest conditions without runtime errors or crashing although the prototype was unable to be evaluated in a real world tropical rainforest.

The results of the testing indicate that the prototype system's sensors have a comparative accuracy to a similar commercially available device but was shown to be inconsistent with the established weather services, and test results included anomalous data. Anomalous data was researched to be filtered in the ARGOS satellite system and appeared to be filtered from the two comparative weather services as well. Perhaps this is a necessary function of a system.

The limited testing conducted within the time available was comparative to the time spent on the other phases of the project but is only considered adequate enough to point to the accuracy of the system which shows that recycled devices and free open source software could potentially be used as a viable alternative in comparison to more sophisticated, expensive academic systems. This was also shown to be possible for bio acoustic audio data by the Rainforest Connection RIS as researched in the literary review and understood from my internship.

Following the phases of an adapted rapid prototyping iterative software engineering project methodology meant that a complete prototype could be delivered and the beginnings of improvements based from the findings of the conducted unit, integration, system, user and acceptance testing can be outlined and planned for and to be implemented in future iterations.

Key aims of the project to reduce the cost and increase the accessibility of an RIS were addressed and are evaluated in section '6.3 The Cost and Accessibility of the Prototype System'. One of the most popular and available recycled smartphones, the Samsung S4 was used for the hardware of the input component purchased for ~$40 (~£30), well below the researched cost of the ARGOS satellite, VHF, GPS and GSM wildlife collars and other systems and towards the lowest cost of recycled fitness activity trackers, smartphones and tablets, with only the recycled fitness tracker researched to be cheaper.

The recycled solar panel, tablet and power bank incorporated from a previous interest, had been previously tested in tropical rainforest environments in Central and South America but not shown formally in this project to be operational in the field. With the right operation and management of the solar panel and battery reserve, the in-the-field free open source Linux laptop can potentially run the central, output and input components without power interruption although the prototype system necessitates an Internet connection to transfer and network its information.

The battery optimisation of the input component was not completed and locally stored data is serialised as Java objects to the devices storage and not the removable micro SD card as plaintext or in .CSV format.

The accessibility to the prototype RIS was evaluated to be greater than the researched academic ARTS system, the commercially available ARGOS system, the Rainforest Connection system and the research SMART collar project that necessitates conderiable infrastructure, high initial and running costs, or are

commercially unavailable, project specific or their source code not available. Cloning the Github project page and purchasing the recycled input component hardware and the only barriers of entry to the system. The central component can run on any computer with Java and Docker installed and a 8000 port forwarded.

The Aims of the project are considered to have been met, but at an unsubstantiated and not comprehensively tested level of degree and only at a very initial stage of the prototype's development.

Perhaps the current state of the project can be more accurately or honestly described as being at a good starting point for future work or future iterations, improvements and developments based on limited testing and limited user feedback but grounded in relatively substantial background research and formalised requirements.

The rapidly developed prototype shows initial viability but early iteration weaknesses. A refined future prototype could make use of the 80:20 where a focus on developing 20% of its current state may produce 80% of the system's total value.

7.1 Current limitations

Currently the prototype system is a relatively well grounded, extendable, low cost, easily accessible and shown to be viable and practical to use in artificial tropical rainforest environments to collect, network, store and visualise field collected sensor data. The system can collect GPS, times and dates, temperatures and humidity sensors readings from different locations for different data collection sessions.

Data can be stored locally or networked to a central server and database where it can be accessed with the web-based interactive 3D globe dashboard and together with the complete IS structure that creates a feedback loop that can be used to analyse the collected field data, the system's performance and the accuracy of the system and its sensors.

The system is currently limited by the amount of sensors it implements, their understood accuracy and the degree to which it has been tested. Although the testing conducted could be seen as adequate for the first release of a rapidly developed prototype developed quickly to base future developments on the early identification of early strengths and weaknesses.

The project's foundational research, formalised requirements and limited and informal user evaluations are reasonably broad and detailed to provide an outline for the future improvements and developments needed that can be used to work towards increasing the alignment of the system, its usefulness and the value the system provides to potential users of the prototype RIS to the expectations of its users and the projects aims.

The limited breadth prototype serves its purpose to illustrate its concept and the test results indicate the level of its performance. This early stage prototype can be used to validate the current progress of the project.

The usefulness of the system is greatly limited by the currently small set of sensors that have been implemented, particularly as the smartphone used was researched to have the most inbuilt sensors of almost any phone available (gsmarena, 2013).

Additionally the software configurations and control of the tracking settings is not present in the current UI. Examples include, being able to increase or decrease timing intervals, the accuracy of the GPS (and battery consumption) and which sensors to record in the session.

Telecommunications and Internet access is limited in remote areas of the rainforest so it necessitates storing data locally in the input component until in a less remote area.
A shortcoming of the prototype system is that the locally stored data is written as serialised objects to the device's storage and not to a removable SD card that would be possible to be recovered if the device became damaged.

During the field and lab testing I attempted to obtain the locally stored data from the input component smart phone which revealed that using serialised objects to store the data was significantly less convenient that using plain text or writing the data to a .CSV format file.

Similarly this would not be required if the system could network data via satellite global communication.

Another considerable limitation of the system is the input component design is device specific to the Samsung Galaxy S4 smartphone. Although only a small number of cell phones were researched to contain the requirement of a built in humidity sensor, the input components design could be modified to be installable on a greater range of possible smartphones with an adequate number of built in sensors with a selectable UI list of sensors to record for a data tracking session.

The central server and database component and output presentation are not limited in the same way and can run on any computer that is able to install compatible versions of Docker and Java used in the system and was additionally shown to be able to run on a very low cost and basic recycled in-the-field Linux laptop previously tested to work well in real world tropical rainforest environments.

The system can be used to collect data offline but necessitates an Internet connection to network the data to the server, although data can be manually entered with SQL directly into the database if it can be deserialized.

The testing of the system was also limited in that it did not test multiple clients collecting and networking field data at the same time. It is expected that issues of transactions from multiple clients could pose transactional problems within the server and database although only write and read functionality is currently implemented and additional error handling code was written to guard against non-unique ID's for Tracks and Points entering the database this is still considered a current limit of the project.

The same is true of the many output componentes retrieving data at the same time, but as only read functionality is currently implemented it is unlikely to pose many errors although this limited testing in this area is another limitation of the project.

The output presentation component only displays the most basic customization of the data being viewed but hints at the possibilities and breadth of a full system implementation. The visualisation and presentation of the collected data can be explored in a variety of different ways including charts, graphs and infographics which are not currently implemented.

The incorporation of supporting contextual tropical rainforest information that was programmed into the database as test data but is not accessible from the output component, or the server's API, also limits the output components potential to be used for data analysis.

The output component can only draw polylines connecting 2 GPS locations resulting in a double line connecting Points on the map. This is currently a significant limitation of the component.

It is likely that the accuracy of the data is only as good as the accuracy of the sensors built into the hardware used. The field experiments showed the viable operation of the system with some limiting anomalous data being unfiltered from entering the system. And the lab testing showed accuracy of the temperature and humidity sensors that is comparable to a commercial device with the same implementation of sensors.

Inconsistencies with official online weather services are still not well understood as many factors could explain the variation in sensor readings between the prototype and the comparative sources for temperature and humidity.

The accuracy of the GPS ranged from exact to within and less than the variation of the accuracy of the ARGOS system (150 metres) researched in the literary review. Another current limitation of the system is the understanding of why the GPS accuracy varied so much in testing and showed a very similar distance and direction deviation from the actual locations of the devices current location.

The quality of the data remains uncertain but the results of testing indicate reasonable accuracy with only some anomalous results. The importance and need for data validation and filtering before data is recorded in the system is important to implement in the next iteration. Filtering data before it is stored in the system would also improve the performance of the JDBC interactivity with the database and help to improve the accuracy and consistency of the data stored in the database.

Limited testing was performed on the consistency of the data within the database particularly with large data set volumes.

Only a subset of the complete set of hardware sensors are implemented. Implementing a greater horizontal breadth of features in the prototype and their vertical depth of functionality would develop the prototype towards a complete system. This would include expanding the database, the processing and serving of data from the database, the input component UI and the data's visualisation in the webpage.

With a more comprehensive capturing of data a more complete measurement of certain aspects of the rainforest can be collected. Incorporating the camera and microphone hardware into the prototype could significantly improve the data collection function of the system.

Another main improvement that could be made in this project is a more accurate understanding of the actions, scenarios, and use-cases of capturing data from a tropical rainforest environment with an RIS by more closely working with professionals, academics and enthusiasts conducting tropical rainforest research.

Revisiting an early stage in the project and writing a test specification for the project based on the requirements could help to measure its success from the start.

Finally, an objective of the project was to test the viability of free and open source software tools to implement the system but in retrospect, many vendor classes were imported without comprehensively researching their licence agreement for use. It is also possible that a licensable version of Java was used to compile the project's Java code. A less limiting choice would be to use licence free versions of libraries and the Java JDK that could help the future of the project avoid vendor lock-in as advised in the research of the literary review.

Evaluation

I am very pleased with my project, it is my first attempt at an academic project of this level and it rekindled my fascination to understand how technology can be applied to research the world's tropical rainforests.

I am very much looking forward to conducting future (improved) projects based on what I have learnt and explored.

It was the first time I have designed and programmed a complete system. It was also the first time I applied a project methodology to a large scale project. I think these firsts show in the often revised processes and inaccuracies.

Several tasks in the project were less documented and in my opinion there may be a disproportionate balance of work weighted in the literary review, design and implementation sections with less comprehensive testing and user evaluation activities.

Some of these limitations can be attributed to the project being conducted during a global pandemic and possible because the project was conducted as a self study project. But there was also a planned parallel aim right from the beginning of the project, before it even started, for it to be a vehicle to familiarise myself with technologies and project processes that would be expected of me in a professional environment. And in many ways the exploration of these processes became a focus.

However, I received very considerate feedback for my preliminary project report and attempted to improve the alignment of my project to be a more scientific based project with a clear aim, clear execution and clear way to measure if the aims had been achieved. I began to really enjoy the scientific aspect of this alignment and as a result I am really enthusiastic about conducting more, but improved, scientific academic projects based on what I have learnt from this one.

I think that the scientific aims at the start of this project were too general, mostly due to unfamiliarity, and I also think this report could be considerably more concise. When reviewing some of the academic journals and articles for the reference section I realised that much more comprehensive and sophisticated systems and projects can be just as easily articulated in considerably fewer words with a greater familiarity and comfort with the processes and formats of scientific academic project writing. Before my next project I plan to read more academic work and even attempt to use contemporary projects as starting points in future projects.

The understanding and use of Agile project process concepts could help in this respect to avoid detail, documentation and to remove processes unlikely to add value.

A difficulty I faced in this project was to accurately define a benchmark of other RIS and FIS systems so that the implementation of this prototype could be measured against. I attempted to frame the project with a focus on cost, accessibility, usefulness, value and ease of use in a way that the achievement of these areas could be evaluated and compared scientifically and the conclusions from the results clear and explicit. And then attempted to present them clearly in an academic project context following the advice of my PPR feedback so that the aims of the project could be measured to really have been achieved.
Establishing more visible metrics for success and progress would have helped the project in this area.

I think the aims of cost, accessibility and ease of use for a prototype were well considered but maybe from only a very educational starting point for a useful RIS.
I think the adoption of the rapid prototyping methodology, a mix of iterative and waterfall development and principled software engineering designs were the greatest strengths of the project making it possible to complete a functioning prototype of limited scope within the time available that can also be easily extended, or improved, in future iterations based on the early identified strengths and weaknesses from testing, user evaluation and feedback. I think these approaches make software easier to design, implement, work with and realign to the requirements of a project.

I really worked hard on this project and its report but I think its level of detail and focus may indicate more of an exploration and understanding of how to research, design, implement, test, evaluate, report and to conduct an academic scientific research project and in general connect where a lot of different technical aspects crossover. The only shortcut I can think of for this learning process is to learn from the previous work and projects of others which is a familiarity I plan to achieve next. The project has been a very illuminating experience and I think this project and its report will be very useful for me in the future. A future which I am looking forward to.

Appendix A: Source code

A.0 Input Component

RainforestApp/app/src/main/res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    tools:layout_editor_absoluteX="16dp"
    android:layout_margin="0dp"
    tools:layout_editor_absoluteY="16dp"
    android:background="@drawable/appbackground" >

    <TextView
        android:id="@+id/message_text"
        android:layout_width="177dp"
        android:layout_height="34dp"
        android:layout_marginBottom="12dp"
        android:fontFamily="serif"
        android:text=" "
        android:textColor="#000000"
        android:textColorHighlight="#FFFFFF"
        android:textColorHint="#FFFFFF"
        android:textSize="15sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.626"
        app:layout_constraintStart_toEndOf="@+id/sendButton" />

    <TextView
        android:id="@+id/humidity_sensor_text"
        android:layout_width="306dp"
        android:layout_height="55dp"
        android:text=" "
        android:textColor="#ffffff"
        android:textSize="20sp"
        android:textStyle="bold|italic"
        app:layout_constraintBottom_toTopOf="@+id/deleteButton"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.161"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <TextView
        android:id="@+id/temperature_sensor_text"
        android:layout_width="343dp"
        android:layout_height="135dp"
        android:layout_marginBottom="28dp"
        android:text=" "
        android:textColor="#ffffff"
        android:textSize="95sp"
        android:textStyle="bold|italic"
        app:layout_constraintBottom_toTopOf="@+id/humidity_sensor_text"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.235"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/location_text"
    app:layout_constraintVertical_bias="1.0" />

<TextView
    android:id="@+id/location_text"
    android:layout_width="300dp"
    android:layout_height="wrap_content"
    android:text="Gps 00.00000, 00.00000\nTime 00:00:00"
    android:textColor="#ffffff"
    android:textSize="25sp"
    android:textStyle="bold|italic"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.153"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />

<Button
    android:id="@+id/startButton"
    android:layout_width="150dp"
    android:layout_height="55dp"
    android:layout_marginBottom="4dp"
    android:text="Start Tracking"
    android:textColorHint="#AE6868"
    android:backgroundTint="@android:color/holo_blue_bright"
    app:layout_constraintBottom_toTopOf="@+id/stopButton"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.065"
    app:layout_constraintStart_toStartOf="parent" />

<Button
    android:id="@+id/stopButton"
    android:layout_width="150dp"
    android:layout_height="55dp"
    android:layout_marginBottom="16dp"
    android:enabled="false"
    android:textColorHint="#AE6868"
    android:backgroundTint="@android:color/holo_blue_bright"
    android:text="Stop Tracking"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.065"
    app:layout_constraintStart_toStartOf="parent"
    tools:ignore="MissingConstraints" />

<Button
    android:id="@+id/sendButton"
    android:layout_width="150dp"
    android:layout_height="55dp"
    android:layout_marginBottom="136dp"
    android:text="Send data to Server"
    android:textColorHint="#AE6868"
    android:backgroundTint="@android:color/holo_blue_bright"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.065"
    app:layout_constraintStart_toStartOf="parent"
    tools:ignore="MissingConstraints" />

<Button
    android:id="@+id/deleteButton"

    android:layout_width="150dp"
    android:layout_height="55dp"
    android:layout_marginBottom="4dp"
    android:text="Delete Files"
    android:textColorHint="#AE6868"
    android:backgroundTint="@android:color/transparent"
    app:layout_constraintBottom_toTopOf="@+id/sendButton"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.065"
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="MissingConstraints" />

    <!--    for future iteration
   <TextView
      android:id="@+id/sensor_list"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="4dp"
      android:text="(Place Holder Sensor List)"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintTop_toBottomOf="@+id/temperature_sensor_text" />

    <ImageView
      android:id="@+id/tracking_image"
      android:layout_width="100dp"
      android:layout_height="95dp"
      android:layout_marginTop="464dp"
      android:visibility="invisible"
      app:layout_constraintBottom_toTopOf="@+id/startButton"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintHorizontal_bias="0.787"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent"
      app:layout_constraintVertical_bias="0.0" />
   -->


</androidx.constraintlayout.widget.ConstraintLayout>
```

RainforestApp/app/src/main/res/values/strings.xml

```
<resources>
    <string name="app_name">Rainforest Sensor Data Tracking</string>
    <string name="error_no_sensor">No sensor</string>
    <string name="humidity_sensor_text">"\n  Humidity %1$.2f"</string>
    <string name="temperature_sensor_text">" %1$.0f°C "</string>
    <string name="location_text">" %1.5f, %2$.5f\n %3$tr\n"</string>
    <string name="no_location">no_location</string>
    <string name="start_tracking">Start Tracking</string>
</resources>
```

RainforestApp/app/src/main/java/com/markstart/rainforest/MainActivity.java

```
package com.markstart.rainforest;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Context;
import android.content.DialogInterface;
```

```
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.location.LocationCallback;
import com.google.android.gms.location.LocationResult;
import com.markstart.rainforest.client.TrackClient;
import com.markstart.rainforest.dataStorage.DataStorageEngine;


// " any interface is better than no interface " Sensors doesn't need an interface ! All ready provided by android ?
// " any Architecture decisions are better than none " -UoL Study Guide 'Software Engineering'.


public class MainActivity extends AppCompatActivity {

    private Context context = this;
    protected static LocationCallback mLocationCallback;

    private Tracker tracker;
    private DataStorageEngine dse;
    private TrackClient trackClient;

    private boolean mTracking = false;

    protected static TextView mSensorHumidityTextView;
    protected static TextView mSensorTemperatureTextView;
    public static TextView mMessageTextView;
    private TextView mLocationTextView;
//  private TextView mSensorListTextView;
//  private ImageView mAndroidImageViewTracking;

    private Button mSendDataButton;
    private Button mStartButton;
    private Button mStopButton;
    private Button mDeleteButton;



    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        dse = new DataStorageEngine(context);

        mSensorHumidityTextView = (TextView) findViewById(R.id.humidity_sensor_text);
        mSensorTemperatureTextView = (TextView) findViewById(R.id.temperature_sensor_text);
        mLocationTextView = (TextView) findViewById(R.id.location_text);
        mMessageTextView = (TextView) findViewById(R.id.message_text);
//      mSensorListTextView = (TextView) findViewById(R.id.sensor_list);
//      mAndroidImageViewTracking = (ImageView) findViewById(R.id.tracking_image);

        mStartButton = (Button) findViewById(R.id.startButton);
        mStopButton = (Button) findViewById(R.id.stopButton);
        mSendDataButton = (Button) findViewById(R.id.sendButton);
        mDeleteButton = (Button) findViewById(R.id.deleteButton);


        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                tracker = new Tracker(context);
                mTracking = tracker.startTrackingLocation();
                toggleUI();
```

```
        }

    });


    mStopButton.setOnClickListener(new View.OnClickListener() {
        @RequiresApi(api = Build.VERSION_CODES.O)
        @Override
        public void onClick(View view) {
            mTracking = tracker.stopTracking(dse);
            toggleUI();
        }
    });


    mDeleteButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

            AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
            builder.setTitle("Delete all files : They can't be recovered.");
            builder.setMessage("Are you sure ?");
            builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                    dse.deleteAllTracks();
                }
            });

            builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialogInterface, int i) {
                }
            });

            AlertDialog dialog = builder.create();
            dialog.show();
        }
    });


    mSendDataButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

            trackClient = new TrackClient(context, dse);
            trackClient.sendAllFiles();
        }
    });


    //needs to be at the bottom of onCreate to be able to make periodic location requests
    mLocationCallback = new LocationCallback() {
        @Override
        public void onLocationResult(LocationResult locationResult) {

            tracker.setmLastLocation(locationResult.getLastLocation());
            new Sensors(context, tracker);      // starts a thread to catch the sensor values asynchronously and calls tracker to add a new Data Point
            updateUI();
            updateScreenDataValues();

        }
    };

}


private void toggleUI() {
    if (mTracking) {
        mStartButton.setText("Tracking ...");
```

```
        mMessageTextView.setText("");
        mStartButton.setEnabled(false);
        mStopButton.setEnabled(true);
        Toast.makeText(this, " Tracking Started, Saving data every 15 mins ", Toast.LENGTH_LONG).show();

    } else {
        mStartButton.setText("Start Tracking");
        mStartButton.setEnabled(true);
        mStopButton.setEnabled(false);

    }
}


private void updateScreenDataValues () {

    mLocationTextView.setText(
            getString(R.string.location_text,
                    tracker.getmLastLocation().getLatitude(),
                    tracker.getmLastLocation().getLongitude(),
                    tracker.getmLastLocation().getTime()));

}


private void updateUI() {

        int pointCount = tracker.getTrack().getTrackPoints().size();

        showToast(" Data Point Saved !" );
        messageHandler("Data Point: " +  pointCount  + " Saved");

}


public static void messageHandler(String string) {

        mMessageTextView.setText( string );
}



protected void showToast(String string) {
    Toast.makeText(context, string, Toast.LENGTH_LONG).show();
}


protected static void sensorError()  {
    mSensorHumidityTextView.setText("sensor_error");
}


@Override
public void onPointerCaptureChanged(boolean hasCapture) {
}


}




// ------------------------------ code for future development iterations ---------------------------


/*

//  Get List of Sensors.
    List<Sensor> sensorList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

```
        StringBuilder sensorText = new StringBuilder();
        for (Sensor currentSensor : sensorList) {
            sensorText.append(currentSensor.getName()).append(
                    System.getProperty("line.separator"));
        }
        TextView sensorTextView = (TextView) findViewById(R.id.sensor_list);
        sensorTextView.setText(sensorText);


// GPS BUTTON request location permission at runtime , NEEDED ??_____

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String[] permissions, @NonNull int[] grantResults) {
        switch (requestCode) {
            case REQUEST_LOCATION_PERMISSION:
                // If the permission is granted, get the location,
                // otherwise, show a Toast
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    getLocation();
                    startButton.setText("getting location");
                } else {
                    Toast.makeText(this,
                            R.string.location_permission_denied,
                            Toast.LENGTH_SHORT).show();
                }
                break;
        }
    }


*/
```

RainforestApp/app/src/main/java/com/markstart/rainforest/Sensors.java

```
package com.markstart.rainforest;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
import android.widget.Toast;


public class Sensors implements SensorEventListener {

    private boolean haveSensorReadings;
    private boolean haveHumidityReading;
    private boolean haveTemperatureReading;
    private SensorManager mSensorManager;
    private Sensor mSensorHumidity;
    private Sensor mSensorTemperature;
    private float currentHumidity;
    private float currentTemperature;
    private Context sensorContext;
    private Tracker sensorTracker;


    public Sensors(Context context, Tracker tracker) {
```

```
    sensorContext = context;
    sensorTracker = tracker;

    mSensorManager = (SensorManager) context.getSystemService(context.SENSOR_SERVICE);

    // Implement a UI list for all sensors
    mSensorHumidity = mSensorManager.getDefaultSensor(Sensor.TYPE_RELATIVE_HUMIDITY);
    mSensorTemperature = mSensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);

    getSensorDataThread thread = new getSensorDataThread();
    thread.start();
}


private class getSensorDataThread extends Thread {

    private getSensorDataThread() {
        haveHumidityReading = false;
        haveTemperatureReading = false;
        haveSensorReadings = false;
    }

    @Override
    public void run() {

        startSensors();

        while (haveSensorReadings == false) {
            if (haveTemperatureReading && haveHumidityReading) {
                haveSensorReadings = true;
            }
        }

        sensorTracker.addNewDataPointToTrack(currentHumidity, currentTemperature);
        stopSensors();
        return;
    }
}


private void startSensors() {

    if (mSensorHumidity != null) {
        mSensorManager.registerListener( this, mSensorHumidity, SensorManager.SENSOR_DELAY_NORMAL);
    } else {
        MainActivity.sensorError();
    }

    if (mSensorTemperature != null) {
        mSensorManager.registerListener( this , mSensorTemperature, SensorManager.SENSOR_DELAY_NORMAL);
    } else {
        MainActivity.sensorError();
    }

}


private void stopSensors() {
    mSensorManager.unregisterListener(this);
}


@Override
public void onSensorChanged(SensorEvent sensorEvent) {

    int sensorType = sensorEvent.sensor.getType();
    float currentValue = sensorEvent.values[0];

    switch (sensorType) {
```

```
            case Sensor.TYPE_RELATIVE_HUMIDITY:

                currentHumidity = currentValue;

                MainActivity.mSensorHumidityTextView
                        .setText(sensorContext.getResources()
                                .getString(R.string.humidity_sensor_text, currentValue));

                haveHumidityReading = true;



                break;

            case Sensor.TYPE_AMBIENT_TEMPERATURE:

                currentTemperature = currentValue;

                MainActivity.mSensorTemperatureTextView
                        .setText(sensorContext.getResources()
                                .getString(R.string.temperature_sensor_text, currentValue));

                haveTemperatureReading = true;

                break;

            default:
        }

    }



    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {
    }


    public float getCurrentHumidity() {
        return currentHumidity;
    }

    public float getCurrentTemperature() {
        return currentTemperature;
    }


}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/Tracker.java

```
package com.markstart.rainforest;

import android.Manifest;
import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Build;
import android.util.Log;
import android.widget.Toast;
```

```
import androidx.annotation.RequiresApi;
import androidx.core.app.ActivityCompat;

import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.markstart.rainforest.dataStorage.DataStorageEngine;
import com.markstart.rainforest.dataStorage.TrackFileNameCreator;
import com.markstart.rainforest.model.Point;
import com.markstart.rainforest.model.Track;

import java.sql.Timestamp;
import java.util.UUID;


public class Tracker {

    private static final int REQUEST_LOCATION_PERMISSION = 1;
    private FusedLocationProviderClient mFusedLocationClient;
    private Location mLastLocation;
    private Track track;
    private Context trackerContext;


    public Tracker(Context context) {

        track = new Track();
        trackerContext = context;
        mFusedLocationClient = LocationServices.getFusedLocationProviderClient(trackerContext);

    }


    protected boolean startTrackingLocation(){

            // the conditional is automatically suggested by android studio but some of the code follows the google android studio tutorial for
permissions in android sensors.
        boolean startedTracking = false;

        try {

            if (ActivityCompat.checkSelfPermission(trackerContext,
                    Manifest.permission.ACCESS_FINE_LOCATION)
                    != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions((Activity) trackerContext, new String[]
                            {Manifest.permission.ACCESS_FINE_LOCATION},
                        REQUEST_LOCATION_PERMISSION);
            } else {
                mFusedLocationClient.requestLocationUpdates(getLocationRequest(), MainActivity.mLocationCallback, null);
                startedTracking = true;
            }
        } catch (Exception e) {
            startedTracking = false;
        }
        finally {
            return startedTracking;
        }

    }


    private LocationRequest getLocationRequest() {

        // 5000 = 5 seconds. finalize software project requirements for sensor performance / timing  before programming.
        LocationRequest locationRequest = new LocationRequest();
        locationRequest.setInterval(900000);    // 900000 is 15 minutes, good for prototype/ testing : )
        locationRequest.setFastestInterval(900000);
        locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        return locationRequest;
```

```
    }



    @RequiresApi(api = Build.VERSION_CODES.O)
    protected boolean stopTracking(DataStorageEngine dse) {

        boolean stoppedTracking = stopTrackingLocation();
        String trackFileName = TrackFileNameCreator.createFileName(track);
        boolean saved = dse.saveTrackToFile(track, trackFileName);

        if (saved) {
            Toast.makeText(trackerContext, " Tracking File Saved ", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(trackerContext, " Could not save file ", Toast.LENGTH_SHORT).show();
        }

        return stoppedTracking;

    }


    private boolean stopTrackingLocation() {

            try {
                mFusedLocationClient.removeLocationUpdates(MainActivity.mLocationCallback);
            } catch (Exception e ) {
                return true;
            }
            return false;

    }


    protected void addNewDataPointToTrack( float humidity, float temperature ) {

        getTrack().getTrackPoints().add( new Point(
                    UUID.randomUUID(),
                    getTrack().getTrack_id(),
                    (float) getmLastLocation().getLatitude(),
                    (float) getmLastLocation().getLongitude(),
                    new Timestamp( getmLastLocation().getTime()),
                    humidity,
                    temperature)
                );

    }



    public Track getTrack() {
        return track;
    }

    public Location getmLastLocation() {
        return mLastLocation;
    }

    public void setmLastLocation(Location mLastLocation) {
        this.mLastLocation = mLastLocation;
    }


}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/model/Point.java

```
package com.markstart.rainforest.model;

import java.io.Serializable;
import java.sql.Timestamp;
import java.util.UUID;


public class Point implements Serializable {

    private UUID point_id;
    private UUID track_id;
    private float gps_latitude;
    private float gps_longitude;
    private Timestamp point_timestamp;
    private float point_humidity;
    private float point_temperature;


     public Point(UUID point_id, UUID track_id, float gps_latitude, float gps_longitude, Timestamp point_timestamp, float point_humidity, float
point_temperature) {
        this.point_id = point_id;
        this.track_id = track_id;
        this.gps_latitude = gps_latitude;
        this.gps_longitude = gps_longitude;
        this.point_timestamp = point_timestamp;
        this.point_humidity = point_humidity;
        this.point_temperature = point_temperature;
    }


    public String createJsonTimestamp() {

        String extendedTime = point_timestamp.toString() + "+00:00";
        String converted = extendedTime.replace(' ', 'T');

        return converted;
    }

    public UUID getPoint_id() {
        return point_id;
    }

    public void setPoint_id(UUID point_id) {
        this.point_id = point_id;
    }

    public UUID getTrack_id() {
        return track_id;
    }

    public void setTrack_id(UUID track_id) {
        this.track_id = track_id;
    }

    public float getGps_latitude() {
        return gps_latitude;
    }

    public void setGps_latitude(float gps_latitude) {
        this.gps_latitude = gps_latitude;
    }

    public float getGps_longitude() {
        return gps_longitude;
    }
```

```
    public void setGps_longitude(float gps_longitude) {
        this.gps_longitude = gps_longitude;
    }

    public Timestamp getPoint_timestamp() {
        return point_timestamp;
    }

    public void setPoint_timestamp(Timestamp point_timestamp) {
        this.point_timestamp = point_timestamp;
    }

    public float getPoint_humidity() {
        return point_humidity;
    }

    public void setPoint_humidity(float point_humidity) {
        this.point_humidity = point_humidity;
    }

    public float getPoint_temperature() {
        return point_temperature;
    }

    public void setPoint_temperature(float point_temperature) {
        this.point_temperature = point_temperature;
    }

}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/model/Track.java

```
package com.markstart.rainforest.model;


import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;


public class Track implements Serializable {

    private final UUID track_id;
    private List<Point> track_points;
//  DataBase Attributes for future iteration to collect more data
//  private String track_name;
//  private String track_description;


    public Track() {
        this.track_id = UUID.randomUUID();
        track_points = new ArrayList<>();
    }


    public UUID getTrack_id() {
        return track_id;
    }

    public List<Point> getTrackPoints() {
        return track_points;
```

```
    }

}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/dataStorage/TrackFileNameCreator.java

```
package com.markstart.rainforest.dataStorage;

import com.markstart.rainforest.model.Track;


public final class TrackFileNameCreator {

    private TrackFileNameCreator(){}

    public static String createFileName ( Track track ) {
        return track.getTrack_id() + ".ser";
    }

}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/dataStorage/DataStorageEngine.java

```
package com.markstart.rainforest.dataStorage;

import android.content.Context;
import android.os.Environment;
import android.util.Log;
import android.widget.Toast;

import com.markstart.rainforest.MainActivity;
import com.markstart.rainforest.model.Track;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;


public class DataStorageEngine {

    private File tracksDirectory;
    private Context dseContext;


   public DataStorageEngine(Context context) {

       dseContext = context;

       tracksDirectory = new File(Environment.getExternalStorageDirectory() + File.separator + "trackFiles");
```

```
    if (!tracksDirectory.exists()) {
        tracksDirectory.mkdir();
    } else {
        Log.d("DSE","DID NOT MAKE A FILE FOLDER *************");
    }

}



public boolean saveTrackToFile(Track track, String fileName) {

    boolean writeSuccessful = false;
    FileOutputStream fs = null;

    if (trackHasAtLeastOnePoint(track)) {

        try {
            fs = new FileOutputStream(new File(tracksDirectory + File.separator + fileName));
            ObjectOutputStream oos = new ObjectOutputStream(fs);
            oos.writeObject(track);
            writeSuccessful = true;
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (fs != null) {
                try {
                    fs.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        return writeSuccessful;
    } else {
        return false;
    }
}


private boolean trackHasAtLeastOnePoint(Track track) {

    if (track.getTrackPoints().size() > 0) {
        return true;
    } else {
        Toast.makeText(dseContext, " Track needs > 0 Data Points ", Toast.LENGTH_LONG).show();
        return false;
    }

}


public Track getTrackFromFile( String fileName) {

    Track track = null;
    FileInputStream fs = null;

    try {
        fs = new FileInputStream(new File( tracksDirectory + File.separator + fileName));
        ObjectInputStream ois = new ObjectInputStream(fs);
        track = (Track) ois.readObject();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
```

```
        return track;
    }


    public void deleteAllTracks() {

        Boolean deletedFiles = false;
        String[] fileList = allFilesList();

        for (String file : fileList) {
            File f = new File(tracksDirectory + File.separator + file);
            deletedFiles = f.delete();
        }

        if (!deletedFiles) {
            Toast.makeText(dseContext, " There are no files to delete. ", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(dseContext, " Files Deleted ", Toast.LENGTH_LONG).show();
        }

    }


    public ArrayList<Track> getAllTracksFromDisk() {

        ArrayList<Track> tracks = new ArrayList<>();
        String [] fileList = allFilesList();

        for (String file: fileList ) {
              tracks.add(getTrackFromFile(file));
        }

        return tracks;

    }


    private String [] allFilesList() {
        File tracksDir = new File(String.valueOf(tracksDirectory));
        String [] fileList = tracksDir.list();
        return fileList;
    }



}
```

RainforestApp/app/src/main/java/com/markstart/rainforest/client/TrackClient.java

```
package com.markstart.rainforest.client;

import android.content.Context;
import android.util.Log;
import android.widget.Toast;

import com.markstart.rainforest.MainActivity;
import com.markstart.rainforest.dataStorage.DataStorageEngine;
import com.markstart.rainforest.model.Point;

import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
```

```
import com.loopj.android.http.*;
import com.markstart.rainforest.model.Track;

import org.json.JSONException;
import org.json.JSONObject;

import cz.msebera.android.httpclient.Header;
import cz.msebera.android.httpclient.entity.StringEntity;
import cz.msebera.android.httpclient.message.BasicHeader;
import cz.msebera.android.httpclient.protocol.HTTP;


public class TrackClient {

    private static String url = "http://markstart.duckdns.org:8000/tracks";
    private static Context contextClient;
    private DataStorageEngine dseClient;


    public TrackClient ( Context context, DataStorageEngine dse ) {

        contextClient = context;
        dseClient = dse;

    }


    public void sendAllFiles() {

        MainActivity.messageHandler("Connecting ...");
        Toast.makeText(contextClient, " Connecting to server ... ... ", Toast.LENGTH_LONG).show();

        ArrayList<Track> tracks = dseClient.getAllTracksFromDisk();

        for (Track track : tracks) {
            for (Point point : track.getTrackPoints()) {
                sendJsonPoint(point);
            }
        }

    }


    private static void sendJsonPoint( Point point) {


        AsyncHttpClient client = new AsyncHttpClient();
        JSONObject json = new JSONObject();
        StringEntity se = null;

        try {
            json.put("point_id", point.getPoint_id());
            json.put("track_id", point.getTrack_id());
            json.put("gps_latitude", point.getGps_latitude());
            json.put("gps_longitude", point.getGps_longitude());
            json.put("point_timestamp", point.createJsonTimestamp());
            json.put("point_humidity", point.getPoint_humidity());
            Log.d("client H", String.valueOf(point.getPoint_humidity()));
            json.put("point_temperature", point.getPoint_temperature());
            Log.d("client T", String.valueOf(point.getPoint_temperature()));


            se = new StringEntity(json.toString());

        } catch (JSONException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
```

```
        try {
            se.setContentType(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        } catch (NullPointerException e ) {
            Log.d("TrackClient", "se.null pointer Exception" );
        }

        client.put(null, url, se, "application/json", new AsyncHttpResponseHandler() {
            @Override
            public void onSuccess(int statusCode, Header[] headers, byte[] responseBody) {

                Toast.makeText(contextClient, " Files Sent ", Toast.LENGTH_LONG).show();
                MainActivity.messageHandler( " Files Sent. ");
            }
            @Override
            public void onFailure(int statusCode, Header[] headers, byte[] responseBody, Throwable error) {

                Toast.makeText(contextClient, " Could not send files ! ", Toast.LENGTH_LONG).show();
                MainActivity.messageHandler( " Files not Sent !" );
            }
        });


    }

}
```

A.1 Central Processing and Data Storage Component

RainforestServer/src/main/java/com/markstart/rainforest/api/PointController.java

```
package com.markstart.rainforest.api;

import com.markstart.rainforest.model.Point;
import com.markstart.rainforest.model.Track;
import com.markstart.rainforest.service.PointService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;



// @CrossOrigin(origins = "http://domain2.com", maxAge = 3600)

@RestController
@RequestMapping("/tracks")
public class PointController {
    private final PointService pointService;

    @Autowired
    public PointController(PointService pointService) {
        this.pointService = pointService;
    }

    @CrossOrigin        //needed for localhost connection from html/.js client
    @GetMapping
    public List<Track> getAllTracks() {
```

```
        return pointService.getAllTracks();
    }

    @CrossOrigin
    @GetMapping(path = "/{track_id}")
    public List<Point> getAllPoints(@PathVariable("track_id") UUID track_id) {
        return pointService.getAllPoints(track_id);
    }

    @RequestMapping(method = RequestMethod.PUT, consumes = MediaType.APPLICATION_JSON_VALUE)
    public void addPoint(@RequestBody Point point) {
        pointService.addPoint(point);
    }
}
```

RainforestServer/src/main/java/com/markstart/rainforest/dao/PointDao.java

```
package com.markstart.rainforest.dao;

import com.markstart.rainforest.model.Point;
import com.markstart.rainforest.model.Track;

import java.sql.Timestamp;
import java.util.List;
import java.util.UUID;

public interface PointDao {


   void insertPointToDB(UUID point_id, UUID track_id, float gps_latitude, float gps_longitude, Timestamp point_timestamp, float
point_humidity, float point_temperature);

   default void insertPointToDB ( Point point ) {
      UUID point_id = point.getPoint_id();
      UUID track_id = point.getTrack_id();
      float gps_latitude = point.getGps_latitude();
      float gps_longitude = point.getGps_longitude();
      Timestamp point_timestamp = point.getPoint_timestamp();
      float point_humidity = point.getPoint_humidity();
      float point_temperature = point.getPoint_temperature();
      insertPointToDB(point_id, track_id, gps_latitude, gps_longitude, point_timestamp, point_humidity, point_temperature);

   }


   List<Track> selectAllTracks();

   List<Point> selectAllPoints(UUID tracks_id);

}
```

RainforestServer/src/main/java/com/markstart/rainforest/dao/PointDataAccessService.java

```
package com.markstart.rainforest.dao;

import com.markstart.rainforest.model.Point;
import com.markstart.rainforest.model.Track;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.Timestamp;
import java.util.List;
import java.util.UUID;


@Repository("postgres")
public class PointDataAccessService implements PointDao {
    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public PointDataAccessService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Override
    public List<Track> selectAllTracks() {
        final String sql = "SELECT track_id, track_name, track_description " +
                "FROM tracks";
        return jdbcTemplate.query(sql, (resultSet, i) -> {
            UUID track_id = (java.util.UUID) resultSet.getObject("track_id");
            String track_name = resultSet.getString("track_name");
            String track_description = resultSet.getString("track_description");
            return new Track(track_id, track_name, track_description);
        });
    }

    @Override
    public List<Point> selectAllPoints(UUID track_id) {
        final String sql = "SELECT * FROM points " +
                "WHERE track_id = '" + track_id + "' ";
        return jdbcTemplate.query(sql, (resultSet, i) -> {
            UUID point_id = (java.util.UUID) resultSet.getObject("point_id");
            UUID tracks_id = (java.util.UUID) resultSet.getObject("track_id");
            float gps_latitude = resultSet.getFloat("gps_latitude");
            float gps_longitude = resultSet.getFloat("gps_longitude");
            Timestamp point_timestamp = resultSet.getTimestamp("point_timestamp");

            float point_humidity = getPointHumidity(point_id);
            float point_temperature = getPointTemperature(point_id);

            return new Point(point_id, track_id, gps_latitude, gps_longitude, point_timestamp, point_humidity, point_temperature);
        });

    }

    private float getPointHumidity(UUID point_id) {
        final String sql = "SELECT sensor_reading FROM point_sensor_readings WHERE point_id = '" + point_id + "' AND sensor_name LIKE
'Humidity'";
        return jdbcTemplate.queryForObject(sql, Float.class);
    }

    private float getPointTemperature(UUID point_id) {
        final String sql = "SELECT sensor_reading FROM point_sensor_readings WHERE point_id = '" + point_id + "' AND sensor_name LIKE
'Temperature'";
        return jdbcTemplate.queryForObject(sql, Float.class);
    }

    private boolean trackInDB(UUID track_id) {
        final String sql = "SELECT EXISTS( SELECT 1 FROM tracks WHERE track_id = '" + track_id + "')";
        boolean exists = jdbcTemplate.queryForObject(sql, Boolean.class);

        return ( exists );
    }

    private boolean pointInDB(UUID point_id ) {
        final String sql = "SELECT EXISTS ( SELECT 1 FROM points WHERE point_id = '" + point_id + "')";
```

```
        boolean exists = jdbcTemplate.queryForObject(sql, Boolean.class);

        return ( exists );
    }

    @Override
    public void insertPointToDB(UUID point_id, UUID track_id, float gps_latitude, float gps_longitude, Timestamp point_timestamp, float
point_humidity, float point_temperature) {
        String humidity = "Humidity";
        String temperature = "Temperature";
        boolean wasAdded = false;

        try {
            if (trackInDB(track_id) == false) {
                jdbcTemplate.update(
                        "INSERT INTO tracks (track_id) VALUES ( ? )",
                        track_id
                );
            }
        } catch (org.springframework.dao.DuplicateKeyException e) {
            System.out.println("caught dulpicate track key ");
        }

        try {
            if (pointInDB(point_id) == false) {
                jdbcTemplate.update(
                        "INSERT INTO points VALUES ( ?, ?, ?, ?, ?)",
                        point_id, track_id, gps_latitude, gps_longitude, point_timestamp
                );

                jdbcTemplate.update(

                        "INSERT INTO point_sensor_readings VALUES ( ?, ?, ?)",
                        point_id, humidity, point_humidity
                );

                jdbcTemplate.update(
                        "INSERT INTO point_sensor_readings VALUES ( ?, ?, ?)",
                        point_id, temperature, point_temperature
                );

                wasAdded = true;

            }

        } catch (org.springframework.dao.DuplicateKeyException e) {
            System.out.println("caught dulpicate point key ");
        }

    }


}
```

RainforestServer/src/main/java/com/markstart/rainforest/datasource/postgresDatasource.java

```
package com.markstart.rainforest.datasource;

import com.zaxxer.hikari.HikariDataSource;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class postgresDatasource {

    @Bean
    @ConfigurationProperties("app.datasource")
    public HikariDataSource hikariDataSource() {
        return DataSourceBuilder
                .create()
                .type(HikariDataSource.class)
                .build();
    }

}
```

RainforestServer/src/main/java/com/markstart/rainforest/model/Point.java

```
package com.markstart.rainforest.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.sql.Timestamp;
import java.util.UUID;

public class Point {

    private UUID point_id;
    private UUID track_id;
    private float gps_latitude;
    private float gps_longitude;
    private Timestamp point_timestamp;
    private float point_humidity;
    private float point_temperature;


    public Point(@JsonProperty("point_id") UUID point_id,
                 @JsonProperty("track_id") UUID track_id,
                 @JsonProperty("gps_latitude") float gps_latitude,
                 @JsonProperty("gps_longitude") float gps_longitude,
                 @JsonProperty("point_timestamp") Timestamp point_timestamp,
                 @JsonProperty("point_humidity") float point_humidity,
                 @JsonProperty("point_temperature") float point_temperature) {
        this.point_id = point_id;
        this.track_id = track_id;
        this.gps_latitude = gps_latitude;
        this.gps_longitude = gps_longitude;
        this.point_timestamp = point_timestamp;
        this.point_humidity = point_humidity;
        this.point_temperature = point_temperature;
    }

    public UUID getPoint_id() {
        return point_id;
    }
```

```
    public void setPoint_id(UUID point_id) {
        this.point_id = point_id;
    }

    public UUID getTrack_id() {
        return track_id;
    }

    public void setTrack_id(UUID track_id) {
        this.track_id = track_id;
    }

    public float getGps_latitude() {
        return gps_latitude;
    }

    public void setGps_latitude(float gps_latitude) {
        this.gps_latitude = gps_latitude;
    }

    public float getGps_longitude() {
        return gps_longitude;
    }

    public void setGps_longitude(float gps_longitude) {
        this.gps_longitude = gps_longitude;
    }

    public Timestamp getPoint_timestamp() {
        return point_timestamp;
    }

    public void setPoint_timestamp(Timestamp point_timestamp) {
        this.point_timestamp = point_timestamp;
    }

    public float getPoint_humidity() {
        return point_humidity;
    }

    public void setPoint_humidity(float point_humidity) {
        this.point_humidity = point_humidity;
    }

    public float getPoint_temperature() {
        return point_temperature;
    }

    public void setPoint_temperature(float point_temperature) {
        this.point_temperature = point_temperature;
    }
}
```

RainforestServer/src/main/java/com/markstart/rainforest/model/Track.java

```
package com.markstart.rainforest.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.UUID;

public class Track {
```

```
    private UUID track_id;
    private String track_name;
    private String track_description;

    public Track(@JsonProperty("track_id") UUID track_id, String track_name, String track_description) {

        this.track_id = track_id;
        this.track_name = track_name;
        this.track_description = track_description;
    }

    public UUID getTrack_id() {
        return track_id;
    }

    public void setTrack_id(UUID track_id) {
        this.track_id = track_id;
    }

    public String getTrack_name() {
        return track_name;
    }

    public void setTrack_name(String track_name) {
        this.track_name = track_name;
    }

    public String getTrack_description() {
        return track_description;
    }

    public void setTrack_description(String track_description) {
        this.track_description = track_description;
    }


}
```

RainforestServer/src/main/java/com/markstart/rainforest/service/PointService.java

```
package com.markstart.rainforest.service;

import com.markstart.rainforest.dao.PointDao;
import com.markstart.rainforest.model.Point;
import com.markstart.rainforest.model.Track;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.UUID;


@Service
public class PointService {
    private final PointDao pointDao;


    @Autowired
    public PointService(@Qualifier("postgres") PointDao pointDao) {
        this.pointDao = pointDao;
    }
```

```
    public List<Track> getAllTracks() {
        return pointDao.selectAllTracks();
    }

    public List<Point> getAllPoints(UUID track_id) {
        return pointDao.selectAllPoints(track_id);
    }

  public void addPoint(Point point) {

        System.out.println("INSERT INTO tracks (track_id) VALUES ('"+ point.getTrack_id() +"');");

        System.out.println("INSERT INTO points VALUES ('"+ point.getPoint_id() + "','" + point.getTrack_id() +"','"+ point.getGps_latitude() + "','"
+ point.getGps_longitude() + "','"+point.getPoint_timestamp()+"');" );

        System.out.println("INSERT INTO point_sensor_readings VALUES ('" + point.getPoint_id() + "', 'Humidity', '" +
point.getPoint_humidity()+"');");
        System.out.println("INSERT INTO point_sensor_readings VALUES ('" + point.getPoint_id() + "', 'Temperature', '" +
point.getPoint_temperature()+"');");


      pointDao.insertPointToDB(point);
    }



}
```

RainforestServer/src/main/resources/db/migration/V1__PointTable.sql

```
CREATE TABLE regions (
    region_name VARCHAR(100) NOT NULL PRIMARY KEY,
    area DECIMAL
);

CREATE TABLE sub_regions (
                sub_region_name VARCHAR(100) NOT NULL PRIMARY KEY,
    area DECIMAL,
    region_name VARCHAR(100) NOT NULL,
    FOREIGN KEY (region_name) REFERENCES regions
        ON DELETE CASCADE
);

CREATE TABLE forests (
                forest_id UUID NOT NULL PRIMARY KEY,
                forest_name VARCHAR(100),
                area DECIMAL,
                forest_description VARCHAR(400)
);

CREATE TABLE forests_sub_regions (
    forest_id UUID NOT NULL,
    sub_region_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (forest_id, sub_region_name),
    FOREIGN KEY (forest_id) REFERENCES forests
        ON DELETE CASCADE,
    FOREIGN KEY (sub_region_name) REFERENCES sub_regions
        ON DELETE CASCADE
);
```

```
CREATE TABLE koppen_types (
   koppen_code VARCHAR(10) NOT NULL PRIMARY KEY,
   koppen_name VARCHAR(100),
   koppen_description VARCHAR(400)
);

CREATE TABLE forests_koppen_types (
   forest_id UUID NOT NULL,
   koppen_code VARCHAR(10) NOT NULL,
   PRIMARY KEY (forest_id, koppen_code),
   FOREIGN KEY (forest_id) REFERENCES forests
      ON DELETE CASCADE,
   FOREIGN KEY (koppen_code) REFERENCES koppen_types
      ON DELETE CASCADE
);

CREATE TABLE tracks (
   track_id UUID NOT NULL PRIMARY KEY,
   track_name VARCHAR(100),
   track_description VARCHAR(400)
);

CREATE TABLE tracks_forests(
   track_id UUID NOT NULL,
   forest_id UUID NOT NULL,
   PRIMARY KEY (track_id, forest_id),
   FOREIGN KEY (forest_id) REFERENCES forests
      ON DELETE CASCADE,
   FOREIGN KEY (track_id) REFERENCES tracks
      ON DELETE CASCADE
);

CREATE TABLE points (
   point_id UUID NOT NULL PRIMARY KEY,
   track_id UUID NOT NULL,
   gps_latitude DECIMAL(10, 7),
   gps_longitude DECIMAL(10, 7),
   point_timestamp TIMESTAMP NOT NULL,
   FOREIGN KEY (track_id) REFERENCES tracks
      ON DELETE CASCADE
);

CREATE TABLE sensors (
   sensor_name VARCHAR(100) NOT NULL PRIMARY KEY,
   sensor_description VARCHAR(400)
);

CREATE TABLE point_sensor_readings (
   point_id UUID NOT NULL,
   sensor_name VARCHAR(100) NOT NULL,
   sensor_reading DECIMAL(6,3),
   PRIMARY KEY (point_id, sensor_name),
   FOREIGN KEY (point_id) REFERENCES points
      ON DELETE CASCADE,
   FOREIGN KEY (sensor_name) REFERENCES sensors
      ON DELETE CASCADE
);


insert into koppen_types values ('Af','Tropical Rainforest',' Usually found within 10 to 15 degrees latitude of the equator, and has at least 60 mm of rainfall every month.');
insert into koppen_types values ('Am', 'Tropical Monsoon', ' With a mean temperature of above 18 C every month, with a wet "Af" season and a "Aw" dry season');
insert into koppen_types values ('Aw/ As', 'Tropical Savanna','With a "Aw" dry winter some "As" months of less than 60mm precipitation');


insert into regions values ('Africa', 30370000);
insert into regions values ('Asia',  44579000);
insert into regions values ('Europe', 10180000);
insert into regions values ('Central America', 201497);
```

```
insert into regions values ('The Caribbean', 2754000);
insert into regions values ('North America', 24709000);
insert into regions values ('Oceania', 8525989);
insert into regions values ('South America', 17840000);



insert into sub_regions values ('New Zealand', 268021, 'Oceania');
insert into sub_regions values ('Australia', 7692000, 'Oceania');


insert into sub_regions values ('Belize', 22966, 'Central America');
insert into sub_regions values ('Costa Rica', 51100, 'Central America');
insert into sub_regions values ('El Salvador', 21041, 'Central America');
insert into sub_regions values ('Guatemala', 108889, 'Central America');
insert into sub_regions values ('Honduras', 112492, 'Central America');
insert into sub_regions values ('Nicaragua', 130375, 'Central America');
insert into sub_regions values ('Panama', 75417, 'Central America');


insert into sub_regions values ('Mexico', 1972550, 'North America');

insert into sub_regions values ('Colombia', 1141748, 'South America');
insert into sub_regions values ('Brazil', 8515767, 'South America');
insert into sub_regions values ('Peru', 1285216, 'South America');
insert into sub_regions values ('Venezuela', 916445, 'South America');
insert into sub_regions values ('Ecuador', 283561, 'South America');
insert into sub_regions values ('Bolivia', 1098581, 'South America');
insert into sub_regions values ('Guyana', 214970, 'South America');
insert into sub_regions values ('Suriname', 163821, 'South America');
insert into sub_regions values ('French Guiana', 83534, 'South America');


insert into sub_regions values ('England', 130279,'Europe');
insert into sub_regions values ('Scotland', 77933,'Europe');
insert into sub_regions values ('Wales', 20779,'Europe');

insert into sub_regions values ('Japan', 377975, 'Asia');



insert into sensors (sensor_name) values ( 'Humidity');
insert into sensors (sensor_name) values ( 'Temperature');




INSERT INTO tracks (track_id) VALUES ('81b0eae3-e9b1-4c6b-95b7-88c99f630aa0');


INSERT INTO points VALUES
('76859c9e-db88-44f9-9756-b458792b5487','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.35739,0.4211113,'2020-08-16 08:08:50.318');
INSERT INTO points VALUES
('4564d42b-ec25-46c6-8978-c7480478d352','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.357464,0.4209414,'2020-08-16 05:38:17.87');
INSERT INTO points VALUES
('39a0d4f6-d768-43f6-8d1e-c6023d87a99d','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.351017,0.5035251,'2020-08-16 07:23:08.82');
INSERT INTO points VALUES
('0316a65b-a192-4db6-a3d1-82da94e595db','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.357597,0.4208168,'2020-08-16 05:53:18.884');
INSERT INTO points VALUES
('c24967af-c489-4972-a676-184bac702c07','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.375523,0.4587482,'2020-08-16 06:38:20.962');
INSERT INTO points VALUES
('5f28d18a-853a-44e8-a747-8b4ee4177afc','81b0eae3-e9b1-4c6b-95b7-88c99f630aa0',51.390102,0.485772,'2020-08-16 06:53:20.917');


INSERT INTO point_sensor_readings VALUES ('4564d42b-ec25-46c6-8978-c7480478d352', 'Humidity', 79.04221);
INSERT INTO point_sensor_readings VALUES ('4564d42b-ec25-46c6-8978-c7480478d352', 'Temperature', 13.599094);
INSERT INTO point_sensor_readings VALUES ('39a0d4f6-d768-43f6-8d1e-c6023d87a99d', 'Humidity', 89.64342);
INSERT INTO point_sensor_readings VALUES ('39a0d4f6-d768-43f6-8d1e-c6023d87a99d', 'Temperature', 14.234207);
INSERT INTO point_sensor_readings VALUES ('c24967af-c489-4972-a676-184bac702c07', 'Humidity', 76.518);
```

```
INSERT INTO point_sensor_readings VALUES ('c24967af-c489-4972-a676-184bac702c07', 'Temperature', 13.765026);
INSERT INTO point_sensor_readings VALUES ('5f28d18a-853a-44e8-a747-8b4ee4177afc', 'Humidity', 63.952595);
INSERT INTO point_sensor_readings VALUES ('5f28d18a-853a-44e8-a747-8b4ee4177afc', 'Temperature', 12.841706);
INSERT INTO point_sensor_readings VALUES ('76859c9e-db88-44f9-9756-b458792b5487', 'Humidity', 96.751396);
INSERT INTO point_sensor_readings VALUES ('76859c9e-db88-44f9-9756-b458792b5487', 'Temperature', 10.698968);
INSERT INTO point_sensor_readings VALUES ('0316a65b-a192-4db6-a3d1-82da94e595db', 'Humidity', 100.0);
INSERT INTO point_sensor_readings VALUES ('0316a65b-a192-4db6-a3d1-82da94e595db', 'Temperature', 10.93708);



INSERT INTO tracks (track_id) VALUES ('bc6307e7-1ef4-43df-8bfb-5d50735206d9');

INSERT INTO points VALUES
('5bff4138-d43f-4ebc-9399-702e3d02820f','bc6307e7-1ef4-43df-8bfb-5d50735206d9',51.502907,-0.3292215,'2020-10-27 11:48:46.686');
INSERT INTO points VALUES
('39487b4e-798a-4192-a264-5f3bc7ebec89','bc6307e7-1ef4-43df-8bfb-5d50735206d9',51.502907,-0.3292221,'2020-10-27 11:48:56.874');

INSERT INTO point_sensor_readings VALUES ('39487b4e-798a-4192-a264-5f3bc7ebec89', 'Humidity', 77.09276);
INSERT INTO point_sensor_readings VALUES ('39487b4e-798a-4192-a264-5f3bc7ebec89', 'Temperature', 17.921455);
INSERT INTO point_sensor_readings VALUES ('5bff4138-d43f-4ebc-9399-702e3d02820f', 'Humidity', 76.64701);
INSERT INTO point_sensor_readings VALUES ('5bff4138-d43f-4ebc-9399-702e3d02820f', 'Temperature', 18.098722);
```

A.2 Output Presentation Component

RainforestServer/src/main/resources/static/rainforestGlobe.html

```
<!DOCTYPE HTML>
<html>

<head>
  <meta name="Rainforest Data Visulization with WebGLEarth" content="index, all" />
  <title>Rainforest Data Visulization with WebGLEarth</title>
  <link rel="stylesheet" type="text/css" href="./rainforestGlobe.css" />
  <script src="http://www.webglearth.com/v2/api.js"></script>
  <script type="text/javascript" src="rainforestGlobe.js"></script>

</head>

<body onload="init()">

  <div id="map"></div>

  <div id="zoomButtons">
    <input type="button" class="transparentButton" value="+" onclick="zoomInOne()"><br>
    <input type="button" class="transparentButton" value="_" onclick="zoomOutOne()"><br>
  </div>

  <div id="animationButtons">
    <input type="button" class="toggleButton" value="Pause Animation" id="animationButton" onclick="pauseAnimation()"><br>
    <input type="button" class="toggleButton" value="Fly to The Hawaii" id="hawaiiButton" onclick="panTo([19.70, -155.55]);"><br>
    <input type="button" class="toggleButton" value="Fly to the other side of the world" id="otherSideButton" onclick="flyToOtherSide()"><br>
  </div>

  <div id="mapSurfaceButtons">
    <input type="button" class="surfaceButton" value="Satellite Map" id="satelliteButton" onclick="applySatelliteMap()"><br>
    <input type="button" class="surfaceButton" value="OpenStreetMap" id="streetButton" onclick="applyStreetMap()"><br>
  </div>

  <div id="trackDataContainer" >
```

```
  <input type="button" class="toggleButton" value="Get All Track Data" id="getdata" onclick="getTrackData()"><br>
  <div id="trackdata"></div>
 </div>

</body>

</html>
```

RainforestServer/src/main/resources/static/rainforestGlobe.css

```
html,
body {
  padding: 0;
  margin: 0;
  background-color: black;
  color: white;
}


#map {
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  background-color: #000;
  position: absolute !important;
}


#getdata {

  border: 1px solid #003f45;
  border-radius: 5px;
  background-color:transparent;

}


#trackDataContainer {

  position: absolute;
  top: 10px;
  left: 10px;
  z-index: 100000;
  overflow-y: auto;

}


#trackdata {
  text-align: left;
  background-color: transparent;
  color: powderblue;
  border: 1px solid transparent;
  outline: none;
  margin: 1px;
}


#mapSurfaceButtons {
  position: absolute;
  top: 10px;
  right: 10px;
  z-index: 100000 ;
```

```
}

.surfaceButton {

  width: 100%;
  background-color: transparent;
  color: #003f45;
  margin: 1px;
  border: none;

}

.surfaceButton:disabled {
  border: none;
  border: 1px solid #003f45;
  border-radius: 5px;
  color: powderblue;
}

.surfaceButton:hover {
  color: #028996;
}

#animationButtons {
  position: absolute;
  bottom: 15px;
  left: 1px;
  z-index: 100000;
}

.toggleButton {
  text-align: left;
  background-color: transparent;
  color: powderblue;
  border: 1px solid transparent;
  outline: none;
  margin: 1px;
}

.toggleButton:disabled {
  color: #003f45;
}

.toggleButton:hover {
  border: 1px solid #003f45;
  border-radius: 5px;
  color: #028996;

}

#zoomButtons {
  position: absolute;
  bottom: 10px;
  right: 10px;
  z-index: 100000;
}

.transparentButton {
  color: white;
  border: none;
  background-color: transparent;
```

```
  border: 1px solid transparent;
  outline: none;
}


.transparentButton:hover {
  border: 1px solid #003f45;
  border-radius: 5px;
}


#animationButton {
  border: 1px solid #003f45;
  border-radius: 5px;
}
```

RainforestServer/src/main/resources/static/rainforestGlobe.js

```
var map;
var animated = true;
var before = null;
var dontAllowTimeElapse;

// const URL = "http://localhost:8000/tracks";        //change url for internet
const URL = "http://markstart.duckdns.org:8000/tracks";



function init() {
  map = WE.map('map', {
    center: [-13, -130],
    zoom: 2,
    unconstrainedRotation: true,
    sky: true,
    dragging: true,
    scrollWheelZoom: true
  });

  applySatelliteMap();
  performAnimation();

}


 function applySatelliteMap() {

    WE.tileLayer('https://webglearth.github.io/webglearth2-offline/{z}/{x}/{y}.jpg', {
      tileSize: 256,
      bounds: [[-85, -180], [85, 180]],
      minZoom: 0,
      maxZoom: 16,
      attribution: 'WebGLEarth Satellite',
      tms: true
    }).addTo(map);

    document.getElementById("satelliteButton").disabled = true;
    document.getElementById("streetButton").disabled = false;

 }


  function applyStreetMap() {
```

```
      WE.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '© OpenStreetMap contributors'
      }).addTo(map);

    document.getElementById("satelliteButton").disabled = false;
    document.getElementById("streetButton").disabled = true;

  }



// Animation method modified from webGLEarth tutorial and thanks to :
https://stackoverflow.com/questions/44347598/webgl-earth-restart-animation-at-current-location
  function performAnimation() {

    toggleFlights(true);

    requestAnimationFrame(function animate(now) {
      if (animated) {
        var c = map.getPosition();
        var elapsed = before ? now - before : 0;

        if (dontAllowTimeElapse) {
          elapsed = 0;
        }

        before = now;
        map.setCenter([c[0], c[1] + 0.1 * (elapsed / 30)]);
        dontAllowTimeElapse = false;
        requestAnimationFrame(animate);
      }
    });
  }


  function pauseAnimation() {
    if (animated) {
      animated = false;
      document.getElementById("animationButton").value = "Start Animation";
      toggleFlights(false);

    } else {
      dontAllowTimeElapse = true;
      animated = true;
      performAnimation();
      document.getElementById("animationButton").value = "Pause Animation";

    }
  }


  function toggleFlights(status) {
    document.getElementById("hawaiiButton").disabled = status;
    document.getElementById("otherSideButton").disabled = status;
  }


  function flyToOtherSide() {
    panTo(antipode(map.getCenter()));
  }


  // From webGLEarth tutorial for inverse cooridantes.
  function antipode(coord) {
    return [-1 * coord[0], coord[1] - 180];
  }

  function setZoom(zoom) {
     map.setZoom(zoom);
  }
```

```
function panTo(coords) {
  map.panTo(coords);
}

function zoomInOne() {
  map.zoomIn();
}

function zoomOutOne() {
 map.zoomOut();
}


function getTrackData() {

    document.getElementById("getdata").disabled = true;

    fetch( URL , {
      method: 'get'
    }).then(function (response) {
      return response.json();
    }).then(function (obj) {
      displayTrackList(obj);
    }).catch(function (err) {
      console.log("error" + err);
    });

}


function displayTrackList(jsonTrackData) {

    var listContainer = document.createElement('div'),
        listElement = document.createElement('ul'),
        listReset = document.createElement('p'),
        numberOfListItems = jsonTrackData.length,
        listItem,
        trackId,
        i;

    listContainer.setAttribute("id", "trackList");
    listReset.innerHTML = '<input type="button" value="Reset List" id="resetButton" class="toggleButton"
onclick="removeElement(trackList); reset()">';

    document.getElementById('trackdata').appendChild(listContainer);
    listContainer.appendChild(listElement);
    listContainer.appendChild(listReset);

    for (i = 0; i < numberOfListItems; i++) {
      trackId = jsonTrackData[i].track_id;
      listItem = document.createElement('li');
      listItem.innerHTML = '<input type="button" value="' + trackId + '" class="toggleButton" onclick="getPointsData(value)"><br>';
      listElement.appendChild(listItem);
    }

  }


  function removeElement(elementID) {
    document.getElementById('trackdata').removeChild(elementID);
  }

  function reset() {
    document.getElementById("getdata").disabled = false;
  }



var trackJson;    // Makes data available for other methods. Best practice ? used in goToPoint()
```

```
function getPointsData(trackId) {

    document.getElementById("getdata").disabled = true;
    removeElement(document.getElementById("trackList"));

    fetch( URL + "/" + trackId, {
        method: 'get'
    }).then(function (response) {
        return response.json();
    }).then(function (obj) {

        trackJson = obj;     // assignment to global variable ?

        displayPointsList(obj);
        displayPointsOnMap(obj);
    }).catch(function (err) {
        console.log("error" + err);
    });

}


function displayPointsList(jsonPointData) {

    var listContainer = document.createElement('div'),
        listElement = document.createElement('ul'),
        listReset = document.createElement('p'),
        numberOfListItems = jsonPointData.length,
        noPointsMessage = document.createElement('p'),
        listItem,
        PointId,
        i;


    listContainer.setAttribute("id", "pointList");
    listReset.innerHTML = '<input type="button" value="Reset List" id="resetButton" class="toggleButton"
onclick="removeElement(pointList); reset()">';

    document.getElementById('trackdata').appendChild(listContainer);
    listContainer.appendChild(listElement);
    listContainer.appendChild(noPointsMessage);
    listContainer.appendChild(listReset);

    if (jsonPointData.status == 500) {
         noPointsMessage.innerHTML = '<p> This track has no point data </p>';
    } else {

        for (i = 0; i < numberOfListItems; i++) {

            jsonItem = jsonPointData[i];
            pointId = jsonItem.point_id;
            listItem = document.createElement('li');
            listItem.innerHTML = '<input type="button" value="Point: ' + [i+1] + '" class="toggleButton" onclick="goToPoint(value)" >';
            listElement.appendChild(listItem);

        }

    }

}


  // use of global trackJson

  function goToPoint(buttonValue) {
    if (animated) {
      pauseAnimation();
    }
    var pointID = buttonValue.substr(buttonValue.length - 1) - 1;
```

```
  panTo( [ trackJson[pointID].gps_latitude, trackJson[pointID].gps_longitude ] );
}

function displayPointsOnMap(json) {

  for (var i = 0; i < json.length; i++) {

    var location = [json[i].gps_latitude, json[i].gps_longitude];
    var marker = WE.marker(location).addTo(map);
    var item = json[i];
    var options = { maxWidth: 120, maxHeight: 100, closeButton: true };
    var pointDateTime = json[i].point_timestamp.split('T');
    var instructions;

    if (i == 0) {

      marker.bindPopup("<span style='font-size:10px;color:black'>" +
        "<b>Start: </b>Click on other points for their point data.<br> Try zooming in if the points are close together.<br><br>" +
        "<b>Point: " + [i + 1] + "</b>" +
        "<br>Date : " + pointDateTime[0] +
        "<br>Time : " + pointDateTime[1].split('.', '1') +
        "<br>Latitude: " + item.gps_latitude +
        "<br>Longitude: " + item.gps_longitude +
        "<br> Humidity: " + item.point_humidity +
        "<br>Temperature: " + item.point_temperature +
        "</span>", options).openPopup();

    } else {

      marker.bindPopup("<span style='font-size:10px;color:black'>" +
        "<b>Point: " + [i + 1] + "</b>" +
        "<br>Date : " + pointDateTime[0] +
        "<br>Time : " + pointDateTime[1].split('.', '1') +
        "<br>Latitude: " + item.gps_latitude +
        "<br>Longitude: " + item.gps_longitude +
        "<br> Humidity: " + item.point_humidity +
        "<br>Temperature: " + item.point_temperature +
        "</span>", options);
    }

  }

  drawLinesBetweenPoints(json);

}

function drawLinesBetweenPoints(json) {

  var pollygonArray = new Array();

  for (var i = 0; i < json.length - 1; i++) {

    var pointArray = [2];
    pointArray[0] = json[i].gps_latitude;
    pointArray[1] = json[i].gps_longitude;
    pollygonArray.push(pointArray);

  }

  var line = WE.polygon(pollygonArray, {
    color: '#03f',
    opacity: 1,
    fillColor: '#ff0',
    fillOpacity: 0.00001,
    weight: 2
  }).addTo(map);

}
```

Appendix B: List of Search Terms and Literary Review Links

Search Terms

Rainforest Information Systems
Forestry Information Systems
Field data collection software
Satellite Communication
Rainforest research technology
GIS
WWF
Tropical rainforest research
Rainforest technology
Animal tracking collars
GPS
Average rainforest conditions
FIS
RIS
Koppen climate classifications
Tropical rainforest organisations

Literary Review Links

2.1 Xprize Rainforest
www.xprize.org/prizes/rainforest
2.2 The Smithsonian Tropical Research Institute Research Project
https://www.researchgate.net/publication/226842427_Tracking_Animal_Location_and_Activity_with_an_Automated_Radio_Telemetry_System_in_a_Tropical_Rainforest
2.3 The Norwegian Polar Institute Research Article
https://polarresearch.net/index.php/polar/article/view/3512/9288
2.4 Wildlife ACT
https://wildlifeact.com/
2.5 SMART Collars Research Project
https://www.researchgate.net/publication/266401454_Instantaneous_energetics_of_puma_kills_reveal_advAntage_of_felid_sneak_attacks
2.6 Rainforest Connection (RFCX) Internship
https://rfcx.org/
2.7 Global Forest Watch
https://www.globalforestwatch.org/
2.8 Regnskogtondet Rainforest Foundation Norway (RFN) Project Guide
https://d5i6is0eze552.cloudfront.net/documents/Publikasjoner/Andre-rapporter/Rainforest-tech-primer.pdf?mtime=20160704134642
2.9 International Tropical Timber Council (ITTO) Summary Report
https://www.itto.int/files/user/pdf/projectformulation/8.pdf
2.10 Open Forests
https://medium.com/openforests/6-principles-that-help-to-design-a-robust-forest-information-system-2fb3177dbe22
2.11 WildLabs
https://www.wildlabs.net/
2.12 Humanitarian Nomad
https://humanitarian-nomad.org/online-selection-tool

Appendix C: References, Figures, Tables

Academic Sources

Kays, R. (2011) 'Tracking Animal Location and Activity with an Automated Radio Telemetry System in a Tropical Rainforest' The Computer Journal, Volume 54, Issue 12. Oxford University Press on behalf of the British Computer Society [Available from]: https://www.researchgate.net/publication/226842427_Tracking_Animal_Location_and_Activity_with_an_Automated_Radio_Telemetry_System_in_a_Tropical_Rainforest

Fuglei, E. & Tarroux, A. (2019) 'Arctic fox dispersal from Svalbard to Canada: one female's long run across sea ice', Research Article, Norwegian Polar Institute, Fram Centre, Norway. [Available from]: https://polarresearch.net/index.php/polar/article/view/3512/9288

Williams, T, M. & Richter, B. (2014) 'Instantaneous energetics of puma kills reveal advantage of field sneak Attacks' Science, Vol 346, Iss. 6205. [Available from]: https://www.researchgate.net/publication/266401454_Instantaneous_energetics_of_puma_kills_reveal_advantage_of_felid_sneak_attacks

Books and Publications

Regnskogfondet - Rainforest Foundation Norway, & The Engine Room (2016) 'Primer on using technology to monitor and share information on rainforest and forests peoples' rights' [Available from]: https://d5i6is0eze552.cloudfront.net/documents/Publikasjoner/Andre-rapporter/Rainforest-tech-primer.pdf?mtime=20160704134642
[Viewed] : 7/08/2020

Simula, M., El-Lakany, H. & Tomaselli, I. (2011) 'META-EVALUATION OF PREVIOUSLY EVALUATED ITTO PROJECTS, Lessons learned & good practices towards sustainable management of tropical forests, Summary Report, 8. Forest information systems' ITTO: La Antigua, Guatemala. [Available from]: https://www.itto.int/files/user/pdf/projectformulation/8.pdf

Blackwell, T. (2007) 'Software engineering algorithm design and analysis', Volume 1, University of London: Study Guide

Murray, D. (2010) 'Interaction Design', University of London: Study Guide

Lewis, D. (2016) 'Database systems: Volume1' University of London: Study Guide

Websites

XPRIZE (2019) 'RAINFOREST', xprize.org,
[Available from] : https://rainforest.xprize.org/prizes/rainforest
[Viewed] : 03/08/2020

Watson, A. (2018) '6 principles that help to design a robust forest Information systems' openforests, medium.com, [Available from]: https://medium.com/openforests/6-principles-that-help-to-design-a-robust-forest-information-system-2fb3177dbe22

[Viewed] : 14/08/2020

Nielsen, J. (1994) 'Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier' Nielsen Norman Group, nngroup.com, [Available from]:
https://www.nngroup.com/articles/guerrilla-hci/
[Viewed] : 22/08/2020

Googler. (2021) 'Advanced Android 03.1: Getting sensor data' Advanced Android Development training course, android.org, [Available from]:
https://codelabs.developers.google.com/codelabs/advanced-android-training-sensor-data/index.html?index=..%2F..advanced-android-training#0
[Viewed] : 30/09/2020

Android. (2021) 'Environment sensors' Android developers documentation, android.org, [Available from]:
https://developer.android.com/guide/topics/sensors/sensors_environment
[Viewed] : 30/09/2020

Humanitarian Nomad. (2020) 'online-selection-tool' humanitarian-nomar.org, [Available from]:
https://humanitarian-nomad.org/online-selection-tool
[Viewed] : 19/08/2020

Dinets, V. (2012) 'Travel in tropical forests, Micro-guide for beginners' dinets.info, ejungletips, [Available from]:
http://dinets.info/ejungletips.htm
[Viewed] : 19/08/2020

Palminteri, S. (2017) 'Location, location, location goes high-tech: Facts and FAQs about satellite-based wildlife Tracking' mongabay.com, [Available from]:
https://news.mongabay.com/2017/05/location-location-location-goes-high-tech-facts-and-faqs-about-satellite-based-wildlife-tracking/
[Viewed] : 21/08/2020

Jones, O. (2012) 'the best way to store GPS coordinates (track) on server' stackoverflow.com, [Available from]:
https://stackoverflow.com/questions/13418066/the-best-way-to-store-gps-coordinates-track-on-server
[Viewed] : 3/010/2020

Peter. (2013) 'Samsung describes what each of the nine sensors on the Galaxy S4 does' [Available from]:
http://blog.gsmarena.com/samsung-describes-what-each-of-the-nine-sensors-on-the-galaxy-s4-does/
[Viewed] : 10/10/2020

Phonegg (2021) 'Cell Phones with Humidity Sensor' phonegg.com, [Available from]:
https://www.phonegg.com/list/302-Cell-Phones-with-Humidity-Sensor
[Viewed] : 11/10/2020

The Telegraph. (2017) '20 bestselling mobile phones of all time' telegraph,co.uk, [Available from]:
https://www.telegraph.co.uk/technology/2016/01/26/the-20-bestselling-mobile-phones-of-all-time/
[Viewed] : 13/10/2020

Bookwalter, J. R. (2013) 'Google Nexus 7 review' techradar.com, [Available from]:
https://www.techradar.com/reviews/pc-mac/tablete Samsung Galaxy S4 was of the 20 most popular smartphones of all times/google-nexus-7-1105336/review/5
[Viewed] : 21/10/2020

Jackson, J. (2017) 'Anker PowerPort Lite 15W Review' outdoorgearlab.com, [Available from]:
https://www.outdoorgearlab.com/reviews/camping-and-hiking/solar-charger/anker-powerport-lite-15w
[Viewed] : 26/10/2020

List of Figures

Fig. 1: The Input Component Java Android client application
Fig. 2: The Central processing and data storage component and system hardware
Fig. 3: The Output presentation component 3D globe dashboard-webpage
Fig. 4: The Track and Point data model mapped on the interactive globe with system collected sensor readings for the GPS location.
Fig. 5: The Input Component recycled hardware
Fig. 6: Comparison of tracking methods (Kays, 2011, p.3).
Fig. 7: A map of the journey (Fuglei & Tarroux, 2019, p.3).
Fig. 8: Interaction design iterative development cycle (Murray, 2010, p.38).
Fig. 9: Making rapid prototyping simpler with vertical and horizontal prototypes (Nielsen, 1994).
Fig. 10: Simple Information System Structure
Fig. 11: Project management plan Gantt Chart
Fig. 12: Simple use-case diagram of the complete system
Fig. 13: System rough sketch
Fig. 14: Input component Class diagram
Fig. 15: Input component Activity diagram
Fig. 16: Input component prototype user interface
Fig. 17: Input component user interface evolution
Fig. 18: Central component webserver and database architecture
Fig. 19: Database systems development process (Lewis, 2016, p.105).
Fig. 20: Refined Entity-Relationship diagram.
Fig. 21: Koppen Climate groups, Af (Rainforest), Am (Monsoon), Aw/As (Savanna, Dry winter/summer), (creative commons).
Fig. 22: Output presentation component
Fig. 23: Prototype RIS hardware
Fig. 24: GPS locations of field experiment 1
Fig. 25: GPS locations of field experiment 2
Fig. 26-34: Input Component Video Walkthrough
Fig. 35: Central processing and data storage component video walkthrough image 1
Fig. 36: Central processing and data storage component video walkthrough image 2
Fig. 37-39 Output presentation component video walkthrough

List of Tables

Table 1: Comparison of tracking collars
Table 2: Comparison of field data collection software applications
Table 3: Comparison of smart devices
Table 4: GPS decimal precision
Table 5: Results of field experiment 1
Table 6: Results of field experiment 2
Table 7: Results of lab test 1
Table 8: Results of lab test 2
Table 9: Generalised comparison of systems