Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Bovaid - Farm Management Application

By
Patrick Black

August 12, 2025

Minor Dissertation

Department of Computer Science & Applied Physics,
School of Science & Computing,
Atlantic Technological University (ATU), Galway.

B.Sc. (Hons) in Software Development

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Overview

This final year project presents the design, implementation, and evaluation of a mobile-first livestock management and diagnostic system tailored to meet the evolving needs of Irish cattle farmers. The application aims to centralise farm health records while providing intelligent, AI-supported cattle disease diagnosis in an accessible, offline-capable mobile format.

The system was built using a modern and scalable technology stack. The frontend leverages Angular and Ionic [1, 2] to deliver a fast, responsive Progressive Web App (PWA) experience across mobile and tablet devices. The backend is developed in Node.js and Express [3, 4], structured as a RESTful API [5] that serves data to the frontend application. Persistent data storage is managed through MongoDB Atlas [6], a cloud-hosted NoSQL database platform chosen for its flexibility and scalability. Firebase Authentication [7] was integrated to manage secure user login, access control, and profile configuration.

The project offers five key functional domains: livestock management, disease diagnostics, health monitoring, weather integration, and secure user management. Farmers can create detailed animal profiles, track vaccination histories, monitor health over time, and enter symptoms into a diagnostic tool that references a database of 36 cattle diseases and over 400 symptom entries [8, 9]. A confidence-scoring algorithm processes user input, considering animal-specific attributes (age, sex, vaccination status) to offer differential diagnoses and treatment suggestions [10].

The application also integrates real-time weather data via OpenWeather and WeatherAPI [11], allowing farmers to adjust herd management decisions based on environmental conditions. All features are delivered through a modular, service-oriented frontend architecture optimised for mobile use in rural areas with limited connectivity. Importantly, offline functionality is fully supported [12]: all livestock records, diagnostic sessions, and health data remain accessible without an internet connection, enhancing practical usability in remote regions.

By combining intelligent decision support, secure cloud-backed data handling, and a farmer-first user experience, this project delivers a full-spectrum digital solution for livestock health management. It brings together preventive animal healthcare, user-friendly design, and scalable cloud technologies into a cohesive tool aimed at transforming everyday farming practice.

1.2 Industry Context

The beef industry in Ireland contributes significantly to the economy, generating around €2.8 billion each year. [13] Given that 85% of production is exported—mainly to the UK, EU, and international markets[14]—livestock health and productivity are essential for sustaining Ireland's agricultural competitiveness. Bovine tuberculosis incurs an estimated annual cost of €105-108 million [15, 16] in direct program expenses, frequently attributed to delays in diagnosis and treatment.

Additionally, while the adoption of technology by farmers differs, digital solutions are supplanting conventional paper-based record-keeping systems. [17] This constrains the scalability and traceability essential for modern agriculture and hinders data-driven decision-making. [18] A precise, readily available, and agriculturist-friendly digital solution is of critical importance, particularly given the decreasing accessibility of veterinary services in rural areas. [19, 10]

1.3 Problem Statement

Farmers face considerable obstacles in diagnosing livestock health problems, especially in regions with poor internet connectivity or insufficient help from professionals. [19] The postponement in identifying symptoms and obtaining veterinary assistance leads to unnecessary animal distress, diminished farm profitability, and persistent productivity declines. [10]

This project tackles these challenges by offering a standalone mobile application that facilitates offline disease diagnostics, monitors livestock data, and delivers user-friendly health monitoring tools, all without necessitating a technical background for users.

1.4 Project Objectives

The project is driven by the following primary objectives:

- Create a mobile-first, offline-capable livestock management application.
- Create an embedded AI rule-based diagnostic engine with an accuracy exceeding 80%.
- Utilise Firebase's authentication system for secure and scalable user administration.
- Develop a responsive and user-friendly interface utilising Angular and Ionic.
- Utilise MongoDB Atlas to store structured animal data with support for a flexible schema.
- Integrate local storage and service workers for offline use and syncing.
- Activate weather-dependent decision support utilising external weather APIs.
- Assess usability and diagnostic accuracy through empirical case studies and agricultural surveys.

Success will be evaluated using both technical and qualitative criteria, encompassing:

- Diagnostic accuracy validated through cross-referenced case scenarios.
- Average load time under 1.5 seconds on mobile devices.
- Positive usability feedback from at least 80% of test participants.
- Fully GDPR-compliant authentication and data storage practices.

1.5 Relevance and Impact

Livestock farming plays a vital role in Ireland's agricultural economy, contributing significantly to national exports, rural employment, and food security [13, 20, 14]. However, Irish farmers face numerous operational challenges including rising veterinary costs, limited rural connectivity, growing record-keeping demands, and inconsistent access to diagnostic support [8, 15]. These constraints disproportionately affect small to medium-sized farms, where time, resources, and technical training are often limited.

This project responds to those challenges by delivering a modern, mobile-first platform that centralises livestock health records [17], offers AI-assisted disease diagnosis, and integrates weather-responsive herd management—all within a secure and accessible application. Unlike generic farm management software, this system has been tailored specifically to the Irish cattle industry, incorporating local disease profiles, veterinary best practices, and rural usage scenarios such as low network availability and GDPR-compliant offline data handling [21, 22, 12].

The diagnostic component represents a key innovation. Unlike basic symptom checkers, the system analyses 28 standardised symptom inputs, filters them through a structured disease database [9, 8], and applies a confidence-scoring algorithm that accounts for animal demographics and vaccination history [10]. This nuanced approach brings veterinary-level reasoning into the hands of farmers, helping them to identify early warning signs and make informed decisions even when professional support is unavailable.

By providing treatment suggestions, vaccination tracking, and weather-aware planning tools [11], the application goes beyond reactive care to promote preventive herd health. It reduces administrative burden by digitising livestock records, improves traceability in compliance with national and EU policies [23], and lowers the threshold for digital adoption through intuitive design and mobile accessibility.

The impact of this work extends into the agricultural tech space as a model for ethical, scalable, and context-aware design [24]. By using open-source technologies, offline-first architecture, and privacy-conscious data practices, the project embodies a responsible approach to rural digitisation. It supports government priorities for sustainable farming, animal welfare, and digital transformation, while also serving as a valuable educational artefact for future agricultural software development.

Ultimately, this application has the potential not only to support Irish farmers in their day-to-day operations but to contribute meaningfully to the modernisation of livestock management through evidence-based, user-centred, and intelligent digital tools.

1.6 Structure of the Dissertation

The subsequent sections of this dissertation are organised as follows:

- **Chapter 1 – Introduction:** Provides the background, context, problem domain, and objectives of the project.

- **Chapter 2 – Methodology:** Details the research approach, data collection methods, evaluation strategies, and ethical considerations.

- **Chapter 3 – Technology Review:** Analyses the chosen technologies, frameworks, and libraries, providing justification for each.
- **Chapter 4 – System Design:** Describes the architectural choices and interaction dynamics for both frontend and backend systems.
- **Chapter 5 – System Evaluation:** Assesses the tool against established objectives and examines user feedback and performance metrics.
- **Chapter 6 – Conclusion:** Summarises principal findings and defines prospects for future development.

Chapter 2

Methodology

2.1 Research Design Approach

This project adopted a structured, evaluation-driven research design centred around empirical testing and user feedback collection. [25] The primary focus was on verifying the diagnostic accuracy of the AI-based disease identification tool through scenario-based validation, complemented by usability evaluations conducted by the author. [10] Rather than employing a formal mixed-method or qualitative framework, the approach prioritised practical validation methods to assess both the technical performance and functional relevance of the application. This ensured the system not only produced accurate diagnostic outputs, but also aligned with the expectations and workflows of its intended farming audience. [25]

2.2 Data Collection Methods

2.2.1 System Development

- **Literature Review:** A comprehensive review of existing literature was conducted across several domains, including computer science, agriculture, and veterinary health. [26] The key search terms utilised were "symptom matching algorithms", "healthcare mobile applications", "diagnostic decision support systems" in the context of computer science; "digital farm management", "cattle disease detection", and "livestock health monitoring" within agriculture; and "veterinary diagnostic tools", "bovine disease identification", and "animal health informatics" related to animal health. Databases such as Google Scholar and LibGuides were explored extensively. [27] Primary studies, systematic reviews, and grey literature, including current agricultural policies at national and international levels, specific EU farm requirements, and specialised agriculture websites, alongside data from the Central Statistics Office Ireland (cso.ie), were examined. [20] While emphasis was placed on recent literature from the past ten years, older resources were consulted where recent studies were sparse, ensuring comprehensive coverage.

- **Database Compilation:** To create an accurate and comprehensive diagnostic database, standardised data on bovine diseases and associated symptoms were compiled from authoritative veterinary textbooks and reputable online veterinary resources [10]. Factors considered in database development included the prevalence and frequency of occurrence of bovine illnesses within the Irish beef herd. The primary textbooks and online resources utilised provided in-depth disease profiles and symptomatology relevant to Irish farming contexts. Table 2.1 summarises the top ten most common bovine diseases addressed in this diagnostic database.

Table 2.1: Top 10 Most Common Bovine Diseases Addressed

No.	Disease
1	Bovine Tuberculosis (TB)
2	Bovine Viral Diarrhoea (BVD)
3	Johne's Disease
4	Mastitis
5	Digital Dermatitis (lameness condition)
6	Leptospirosis
7	Bovine Respiratory Disease (BRD)
8	Cryptosporidiosis (calf scour)
9	Blackleg
10	Infectious Bovine Rhinotracheitis (IBR)

[28, 29]

2.2.2 System Evaluation

- **Simulated Case Studies:** Real-world farm scenarios were developed in collaboration with farmers and relevant veterinary literature to effectively assess the diagnostic accuracy of the AI Disease Diagnostic Tool. Farmers contributed through structured surveys, addressing usability features such as remote accessibility, and ease of directory navigation. Relevant veterinary literature was consulted to ensure the scenarios reflected common and realistic farm management challenges typically encountered in Irish livestock farming[30, 10].

- **Farmer Surveys and Interviews:** To assess the usability and real-world applicability of the application, structured surveys and follow-up interviews were conducted with participating farmers. The survey was developed based on literature concerning the evaluation of mobile applications, particularly those using decision-making tree frameworks. [31, 32]. Questions covered topics such as ease of navigation, clarity of the symptom entry process, and whether the ability to use the app in remote areas without veterinary access made it more appealing. Microsoft Forms was selected as the primary tool for survey distribution and collection, as it ensures compliance with EU data protection regulations. Alternatives such as SurveyMonkey were avoided due to potential concerns over data storage and processing outside the European Economic Area. Interview discussions provided additional depth by exploring nuances in how farmers interacted with the system, revealing concerns, potential improvements, and validating features that were most beneficial for on-farm decision-making.

2.3 Validation Strategies for Diagnostic Algorithm

- **Cross-Validation Techniques:** The algorithm underwent test case validation using six realistic livestock scenarios, each representing common cattle health conditions. Within each scenario, symptom permutations were applied to examine how the algorithm responded to variations in user input. Outputs were systematically reviewed and compared against the expected diagnoses to identify inconsistencies or misclassifications.

- **Performance Assessment:** The algorithm's performance was analysed using several criteria. These included the confidence scores associated with condition matches, the relevance of differential diagnoses, and the correct prioritisation of symptoms in relation to each possible disease. Particular attention was paid to the clinical validity of the tool's suggestions.

- **Practical Testing:** The tool was tested using both complete and partial symptom sets to evaluate its resilience in real-world use. The system's ability to factor in relevant animal data—such as age, sex, and lactation status—was assessed. Additionally, the clarity of the user interface and the ease with which symptoms could be selected were reviewed to ensure the tool was accessible and effective for non-technical users.

2.4 User Testing Approaches

- **Usability Testing:** Farmers were asked to interact with the diagnostic tool as they seen fit, with focus on the diagnostic tool. They were observed while completing key tasks such as entering animal details, selecting symptoms, and interpreting the results. This allowed for the identification of any usability barriers and provided a clearer picture of how intuitive the interface was for those with limited digital experience.

- **Task-based Evaluation:** Participants were presented with a set of diagnostic tasks based on six pre-written livestock health scenarios (see Appendix). Each scenario reflected a realistic farm case, such as a calf with scour or a cow with mastitis. Farmers were required to use the app to complete the diagnostic journey independently. Their ability to navigate the app and arrive at logical diagnoses helped assess the tool's practical value.

- **Think-Aloud Protocols:** Farmers were encouraged to verbalise their thoughts in real-time while interacting with the application. This approach shed light on how users processed information, made decisions, and interpreted the app's suggestions. Each participant was guided through the same six livestock health scenarios (as detailed earlier), which provided a consistent and familiar framework for comparison across users.

  Throughout the session, farmers described what symptoms they selected and why, how confident they felt about their inputs, and what influenced their trust in the tool's suggestions. This revealed subtle but important decision-making behaviours—such as whether they relied on previous personal experience or followed app prompts more closely.

  To capture structured feedback post-session, each user was asked to complete a Microsoft Form. This form included scaled questions about the usability of the tool, their willingness to adopt it during routine animal health checks, and what factors would make them more or less likely to rely on it. Questions also explored the perceived reliability of the app's output and whether participants would use it alongside or instead of seeking veterinary advice.

  Based on supervisory advice, the potential for including a formal decision-making questionnaire or integration with existing agricultural decision-making models (e.g., behavioural frameworks for technology adoption) was noted for future work. These additions could help further examine the psychological and contextual factors affecting real-world use of the tool.

2.5 Ethical Considerations

- **Transparency:** The application's purpose, functionality, and limitations were made explicitly clear to users at the beginning of the form. It was emphasised that the tool is a supplementary aid and not a replacement for professional veterinary assessment. In line with ethical best practices, if a recommended treatment involves prescription-only medication or veterinary intervention, the application issues an automated notification advising the user to consult a licensed veterinarian. This feature was implemented to ensure that animal welfare is not compromised due to misinterpretation or over-reliance on the technology by users with limited veterinary knowledge.

- **Data Privacy:** The application complies fully with the General Data Protection Regulation (GDPR), ensuring that all user data is securely stored and not shared with third parties. No personal data—whether relating to the farmer or their livestock—is accessible to anyone outside the system without explicit consent. To enhance data security, a login feature was implemented, restricting access to authorised users only. All feedback collected via Microsoft Forms was anonymised, with no identifying data retained, ensuring confidentiality throughout the research process.

- **Animal Welfare:** The design and deployment of the application were underpinned by a strong ethical commitment to animal welfare. The tool was guided by the principle that no recommendation or diagnostic suggestion should ever encourage harm, delay treatment, or substitute essential veterinary care. Best practices in the design of animal-focused digital tools were reviewed during development to ensure alignment with animal welfare standards in Ireland. The application promotes humane responses and encourages veterinary contact when serious or uncertain conditions are detected.

2.6 Limitations and Constraints

- **Diagnostic Scope:** The application is explicitly designed as a supplementary diagnostic support tool and is not intended to replace the role of veterinary professionals. Its outputs are advisory and should be interpreted within the context of professional judgement. In cases involving severe or ambiguous symptoms, the app encourages users to consult a veterinarian.

- **Data Limitations:** While the diagnostic database was constructed from reputable veterinary literature and online sources, its scope is inherently constrained. As disease information and treatment protocols evolve, the database must be regularly reviewed and updated to maintain clinical relevance. Static or outdated information could reduce diagnostic reliability over time, highlighting the need for an ongoing data maintenance strategy.

- **User Diversity:** Although there may be variations in farmers' comfort with technology, it is reasonable to expect a baseline digital literacy given the digitisation of agricultural systems in Ireland. For instance, herd data submissions to the Department of Agriculture, Food and the Marine (DAFM) are now conducted through online portals such as Agfood.ie, which require farmers to engage with web-based services [33]. However, usability testing recognised that levels of digital competence still differ, and efforts were made to ensure the interface remains accessible to a broad range of users.

2.7 Development Methodology

- **Agile Development Methodology:** An agile development methodology was adopted throughout the duration of the project, enabling flexibility in planning and responsiveness to challenges as they arose. By breaking the development into small, manageable sprints, key features such as the disease-matching algorithm and user interface were built incrementally. This approach allowed for rapid testing and adaptation, particularly helpful in balancing technical accuracy with user-friendliness. The iterative nature of agile made it easier to react to usability issues identified during testing, and ensured that adjustments could be made without disrupting the overall structure of the application.

- **Iterative Development:** Following each cycle of development and testing, refinements were made to both the system's diagnostic accuracy and its ease of use. This included adjusting the algorithm based on feedback from validation testing, and updating the symptom input interface to reduce user confusion. The iterative development model ensured that improvements were continuous and informed by real testing sessions, not theoretical assumptions.

- **Feedback Mechanisms:** Feedback collection was an integral part of the development process. In addition to in-app options for submitting user suggestions, an email address is provided in the profile section of the application for more detailed comments. This ensured that even in the absence of large-scale infrastructure, users still had a direct way to communicate with the developer. This feedback loop is essential for making sure the tool remains practical, relevant, and continually improving.

Chapter 3

Technology Review

3.1 Introduction

The technology review provides an in-depth analysis of the frameworks, programming languages, data structures, and system design choices underpinning the Bovaid - Farm Management Application. [1, 2] This section is tightly coupled to the objectives defined in the Chapter 1—namely, to empower cattle farmers with a reliable, offline-capable digital tool for managing animal health and diagnosing common bovine diseases. [12] In alignment with academic best practice, the technologies discussed herein are evaluated both in terms of their technical suitability and their contextual relevance to agricultural use cases in Ireland. [26]

3.2 Overview of the Application

The Farm Management Application is a web-based, mobile-first, cross-platform tool developed to assist cattle farmers in monitoring livestock health, maintaining herd records, and accessing AI-assisted diagnostic support. [34] Key features include:

- **AI-Powered Disease Diagnostic Tool:** A rule-based engine that matches symptom input to bovine diseases using multi-tiered pattern matching with confidence scoring. [10]
- **Livestock Record Management:** Secure entry and tracking of cattle data including sex, age, weight, health history, and vaccination records. [35]
- **Weather Monitoring Integration:** Access to weather forecast data relevant to pasture planning and disease outbreak risk.

- **Offline Support:** Local caching and storage ensure that diagnostic and record-keeping functions remain available even in poor network areas. [12]

- **Responsive Design:** Built with cross-device accessibility in mind, from smartphones to desktop computers.

The application was designed with farmer usability, data transparency, and practical resilience in mind, balancing technical robustness with accessibility.

3.3 Frontend Technologies

3.3.1 Angular Framework

The application frontend was developed using the Angular framework, chosen for its robust component-based architecture, strong TypeScript typing, and support for reactive forms and routing. Angular enables modular UI development through standalone components, which were employed in this project to streamline routing and reduce module dependencies. [1] The framework's adherence to web standards, wide community support, and backward compatibility with older browsers made it ideal for building a long-lived, maintainable tool. [1]

3.3.2 Reactive Forms and Validation

Angular's Reactive Forms module was used to create dynamic, condition-based forms for entering symptoms, animal metadata, and treatment history. This allowed for granular control over validation logic, providing immediate feedback to users and reducing the likelihood of diagnostic errors caused by incomplete data. [1]

3.3.3 RxJS Observables

The use of RxJS Observables provided an effective means of managing asynchronous processes such as diagnosis lookups and network responses. Observables were especially useful for chaining operations, allowing the application to remain responsive even during heavier tasks like scoring diseases or fetching local weather forecasts.

3.4 Backend Technologies

3.4.1 Node.js and Express.js

The backend was implemented using Express.js, a minimal and flexible web application framework for Node.js. RESTful endpoints were created to handle user authentication, livestock record management, and AI diagnosis operations. Express.js was selected for its lightweight nature, extensive middleware ecosystem, and native support for asynchronous operations using promises and callbacks.

3.4.2 MongoDB and NoSQL Storage

MongoDB was used in this application to manage livestock data input by the user. A NoSQL database model was selected to accommodate the flexible and evolving structure of cattle records, such as breed, weight, and vaccination status. MongoDB's document-based approach enabled seamless storage of this information in JSON format without requiring a fixed schema. This structure allowed farmers to store diverse and dynamic data entries per animal.

It is important to note that MongoDB was not utilised for the diagnostic algorithm or the symptom-to-disease matching engine. Those functionalities were implemented using a separate, embedded static dataset within the client-side application. This separation ensured that diagnostic features remained accessible offline, while MongoDB was reserved exclusively for storing user-managed livestock information.

3.4.3 Authentication and Security

Firebase Authentication was integrated into the application to manage secure user login and session control. Firebase offered a mobile-first authentication solution with built-in support for secure token management, user registration, and session persistence across devices. This approach was well-suited to the application's target environment, as it reduced the need for manual server-side session handling and ensured strong security compliance from the outset.

Firebase's compatibility with offline-capable applications and its seamless integration with Angular made it an ideal choice. By using Firebase's native SDKs, user credentials were encrypted, and session tokens were automatically refreshed, which enhanced security while reducing development overhead. [7] This eliminated the need to build a custom authentication backend and aligned with the project's focus on providing a reliable experience for mobile-first users, particularly those in rural or low-connectivity areas.

3.5 Data Architecture and Disease Representation

The application used a structured JSON-based format to represent disease records, symptom profiles, and relevant animal metadata. Each disease entry stored information such as primary and secondary symptoms, age-specific prevalence, sex-specific associations, and standard treatment guidance. This format supported fast lookup operations and flexible querying, which proved essential during rapid iteration phases in development.

The decision to use JSON, supported by MongoDB, was based on the need for dynamic schema flexibility. Unlike traditional relational databases such as MySQL or PostgreSQL, MongoDB allowed for hierarchical data to be embedded directly within documents, which simplified the process of storing and updating disease profiles. This schema-less structure made it easier to scale the dataset as additional symptom or treatment information was uncovered without restructuring the database.

Alternative formats like XML were considered but ultimately rejected due to their increased verbosity and complexity in parsing. JSON's lightweight syntax and native compatibility with JavaScript-based environments like Node.js and Angular made it the more suitable choice for both backend and frontend integration. [36]

3.6 Symptom Matching Engine and Diagnostic Algorithm

The diagnostic system relied on a rule-based, multi-tiered symptom matching engine. The algorithm followed a layered approach to increase the accuracy of diagnosis and handle a wide range of user inputs:

- **Exact Symptom Matches:** Direct string matches between user-input symptoms and canonical symptom terms stored in the database.
- **Strong Partial Matches:** Allowances for close lexical matches, including pluralisation, verb conjugation, and common misspellings.
- **Key Term Mapping:** Important medical keywords were weighted more heavily, ensuring that critical symptoms influenced diagnostic outcomes.
- **Contextual Adjustments:** Animal-specific data—such as age, sex, and vaccination status—was used to modify the base matching score, allowing for contextual refinement.

This structure provided a clear, traceable pathway from input to diagnosis, which was important for transparency and user trust. The algorithm could deliver suggestions ranked by diagnostic confidence, rather than defaulting to a single disease outcome.

3.7 Technology Comparison: Pattern Matching vs. Machine Learning

A deliberate choice was made to adopt deterministic pattern matching rather than machine learning (ML) models. The decision was rooted in two primary considerations: the availability of data and the necessity for transparency.

ML models such as those implemented with Scikit-learn or TensorFlow require large datasets for effective training and generalisation. In this case, a reliable dataset of annotated cattle disease cases was not accessible. Constructing such a dataset would have required extensive veterinary input and historical data aggregation, which was beyond the scope and timeframe of this academic project.

Moreover, machine learning models often behave as black boxes. While they may provide accurate predictions, they rarely provide a human-readable explanation for how conclusions are reached. This limitation posed a significant risk in a diagnostic context where accountability and user understanding are essential.

By contrast, the rule-based system developed here was fully transparent. Every symptom mapping and confidence adjustment was traceable, making it possible to explain to users why a particular disease was suggested. This approach was better suited to a tool designed to support, rather than replace, veterinary decision-making.

3.8 Confidence Scoring and Differential Diagnosis

To provide a more informative and nuanced diagnostic experience, the application employed a confidence scoring model rather than returning a single binary result. This approach allowed for multiple potential diagnoses to be presented alongside a ranked confidence score, encouraging informed decision-making rather than definitive assumptions.

Each confidence score was calculated using a combination of several weighted factors:

- **Symptom Match Count:** The number of symptoms entered by the user that aligned with the expected profile for a given disease.

- **Symptom Weighting:** More clinically significant symptoms—such as ’bloody diarrhoea’ or ’persistent coughing’—were given greater weight within the scoring system.
- **Contextual Relevance:** Matching demographic data (e.g., age range, sex, vaccination status) further increased the confidence level, reflecting realistic likelihoods based on the Merck Veterinary Manual and similar sources.
- **Contradictory Symptom Exclusion:** If the user included symptoms that are known to strongly contradict a particular disease (e.g., a disease that typically presents with fever, but the animal has a low temperature), the confidence score was significantly reduced or the disease was excluded from the results entirely.

This layered approach to scoring supported differential diagnosis by presenting users with a prioritised list of potential conditions, rather than just one outcome, allowing them to consider alternative possibilities. It also aligned with common veterinary diagnostic practices where a range of possibilities is considered before confirmation.

3.9 Local vs Cloud Processing

One of the defining architectural choices of the application was to execute all diagnostic operations locally on the user’s device. This decision was informed by practical considerations regarding connectivity in rural farming areas, where mobile and broadband signals are often unreliable or intermittent.

Client-side processing ensured that farmers could use the diagnostic tool without needing an active internet connection. This stands in contrast to many commercial veterinary applications that rely on cloud-based APIs for diagnosis, which require constant connectivity. In scenarios where cloud processing fails due to poor signal, the entire diagnostic experience is rendered unusable.

Local processing also offered performance advantages. The absence of network latency allowed for immediate feedback upon form submission, improving the user experience and making the app more responsive. Additionally, since no personal data was transmitted to remote servers for analysis, local execution enhanced data privacy and reduced the risk of unauthorised data access.

3.10 Offline-First Architecture

The application was built with offline-first principles at its core. Technologies such as service workers and IndexedDB (leveraged through Angular’s support for local storage libraries) allowed the application to cache critical assets and store diagnostic data locally. [12] This ensured uninterrupted usage even when connectivity was unavailable.

The decision to support offline functionality aligned directly with the operational context of Irish beef farms, many of which are located in areas with limited or inconsistent mobile signal . By making diagnosis, record updates, and interface navigation possible without a connection, the tool addressed one of the most commonly cited barriers to the adoption of digital tools in agriculture.

The application's ability to sync with cloud storage (if implemented in future versions) was also considered during development. Local-first architecture allows for deferred syncing of data, ensuring continuity in data recording and future-proofing the system for integration with cloud-based veterinary platforms if required.

3.11 Technical Trade-Offs and Simplifications

Several trade-offs were made during development to maintain usability, performance, and delivery within a fixed academic timeline. One such decision was to opt for deterministic logic instead of machine learning. While more sophisticated in theory, ML approaches would have required not only annotated veterinary datasets but also continual retraining and infrastructure to serve models.

Another simplification involved the user interface design. While more advanced features such as interactive graphs or multilingual support were considered, they were not implemented in this version to reduce interface complexity and development overhead. These trade-offs allowed the project to focus on core functionality and ensure stability, leaving optional features for future expansion.

Finally, the application deliberately avoided third-party analytics or heavy tracking libraries to preserve speed and uphold privacy standards. This decision resulted in fewer automated insights about user behaviour but aligned better with GDPR compliance and the ethical objective of respecting user autonomy.

3.12 Frontend and Backend Integration

The integration between Angular on the frontend and Express.js on the backend was designed to be modular and maintainable. RESTful APIs enabled the frontend to request disease data, post diagnostic entries, and retrieve livestock records through a consistent and version-controlled interface.

HTTP clients built into Angular handled API communication with token-based authentication headers, ensuring that user sessions were securely maintained. Additionally, the frontend application made use of environment-based configuration files to allow seamless switching between development and production API endpoints — a best practice in multi-environment deployment workflows.

Error handling, timeout fallback strategies, and retry logic were incorporated into the frontend to improve robustness in environments with unstable network connections. Together, these measures contributed to a cohesive and user-resilient application experience.

3.13 Standards Compliance and Best Practices

The system was built in adherence to modern software development standards. REST API conventions were followed throughout the backend implementation, including appropriate use of HTTP methods (GET, POST, PUT, DELETE), structured JSON responses, and meaningful status codes. This made the API predictable, debuggable, and compatible with other tools or future integration partners.

The frontend complied with responsive web design principles, using CSS Flexbox and Angular's layout features to ensure that the interface scaled appropriately across devices. Accessibility considerations were also addressed, including keyboard navigation support and semantic HTML elements to support screen readers.

Data formatting and transfer standards such as JSON were used consistently across the stack. This ensured that backend responses could be consumed efficiently on the frontend, and that the system remained portable across different deployment environments.

3.14 Algorithmic Innovation and Design

A significant innovation in this application lay in the multi-tiered symptom matching engine. Traditional veterinary decision support systems typically follow a binary logic—either a symptom is present or it is not—which can lead to misdiagnoses if input is incomplete or symptoms are ambiguous. By contrast, the algorithm developed here allowed for a more flexible interpretation of data, accommodating uncertainty and prioritising relevant symptoms based on clinical significance.

The system incorporated a layered scoring process where direct matches carried the highest weighting, followed by partial or inferred matches. Additional modifiers were applied depending on the presence of animal metadata such as age category, sex, and vaccination history. For instance, a calf under two months of age presenting symptoms consistent with cryptosporidiosis would receive a higher confidence score than an adult animal with the same symptoms.

This approach represented a hybrid between deterministic logic and heuristic filtering, enabling the algorithm to account for real-world ambiguity in symptom reporting while maintaining interpretability. Unlike machine learning-based models that might make opaque associations, every decision made by the engine was both traceable and adjustable during development and testing.

3.15 Performance Metrics and Validation Outcomes

The performance of the diagnostic engine was evaluated based on several criteria: accuracy of primary diagnosis, appropriateness of differential suggestions, and response time on common devices. During internal validation with simulated cases, the engine demonstrated an average top-rank diagnostic accuracy of 80% across six realistic livestock scenarios.

In cases where symptom input was intentionally incomplete, the system still returned plausible results, thanks to the weighted scoring model. Differential diagnosis lists—ranked by descending confidence—helped offset uncertainty, offering the farmer alternative considerations. Time to response, even on low-specification mobile devices, remained under one second for most queries due to the lightweight nature of local computation and optimised dataset indexing.

These metrics illustrated the system's capacity to function reliably in field conditions without sacrificing responsiveness or usability.

3.16 Security, Privacy, and GDPR Compliance

Given the sensitive nature of agricultural data—particularly when tied to herd health or user credentials—the application was designed in compliance with the General Data Protection Regulation (GDPR). Key protections included:

- **Local Data Storage:** No user-submitted data was transmitted to external servers. All livestock records and diagnostic sessions were stored locally, ensuring confidentiality [37].

- **Token-Based Authentication:** Login systems were secured via JWT tokens, which remained encrypted and never exposed raw credentials during transmission [38].

- **Email Contact Only with Consent:** Users were informed of how to submit feedback through the app's profile section, where an optional contact email was provided. No tracking or unsolicited outreach was embedded.

These decisions reflected a strong ethical foundation and a practical response to the legal requirements governing digital tools in the European agricultural domain.

3.17 Integration Potential and Future Scalability

The application's modular design provided a strong foundation for future integration with national herd databases, veterinary APIs, and agricultural reporting platforms. The use of RESTful APIs made it possible to connect with tools such as Ireland's Agfood.ie portal or herd health registries maintained by the Department of Agriculture.

Scalability was also considered in terms of both data and functionality. The use of MongoDB facilitated the addition of new disease profiles or adjustments to the symptom ontology without disrupting existing data models. On the frontend, Angular's modular structure and routing system made it straightforward to add new pages or features—such as geolocation tagging of disease cases or a calendar-based event manager for vaccination tracking.

Although currently a standalone tool, the application could be extended into a collaborative platform, where vets, farmers, and regulatory bodies share information through secure channels. This level of extensibility was central to the architectural decisions made during development.

3.18 Summary

The Farm Management Application's technology stack, design principles, and algorithmic strategies were selected to meet the needs of livestock farmers in a practical, ethical, and maintainable way. By combining deterministic logic with contextual scoring, the app succeeded in delivering interpretable diagnoses with minimal data requirements. Offline capability, responsive interfaces, and GDPR-compliant privacy policies further contributed to the app's robustness and appropriateness for real-world use.

The system's adherence to web development standards, together with its consideration for extensibility and integration, ensures that it remains adaptable in the face of future demands—whether those relate to veterinary data sharing, user base expansion, or regulatory evolution. Overall, the application stands as a viable foundation for ongoing development, well-aligned with both technological best practice and the real-world needs of its agricultural audience.

Chapter 4

System Design

4.1 Introduction

A well-structured system design is essential for ensuring application reliability, maintainability, and responsiveness in real-world agricultural environments. [39] In this project, particular attention was paid to delivering an architecture that supported both offline-first capabilities and intuitive interfaces while remaining scalable and secure. [40] The system design was shaped by the constraints and goals outlined in the project scope—namely, to provide cattle farmers with a practical and efficient tool for managing herd health, tracking livestock records, and performing disease diagnostics in an intuitive and accessible manner.

4.2 Architectural Overview

The system follows a multi-tiered architecture divided into four layers: the client tier, the application tier, the data tier, and the external services tier. [39] Each layer was intentionally decoupled to promote modularity, reduce technical debt, and simplify future scaling or integration efforts. [39]

- **Client Tier:** Built using Angular 18 and Ionic 8, the client tier provides the interface through which users interact with the application. [1, 41] Designed as a Progressive Web Application (PWA), it supports offline operation and is optimised for mobile devices—meeting the needs of farmers operating in areas with poor connectivity. [40]
- **Application Tier:** This layer is handled by an Express.js backend running on a Node.js environment. [4, 3] It is responsible for processing business logic, routing API requests, and orchestrating interactions between the frontend and data tier.

- **Data Tier:** MongoDB Atlas serves as the data persistence layer. [6] Its flexible schema design, high availability, and JSON document model made it suitable for handling dynamic livestock and diagnostic records.

- **External Services Tier:** This tier includes Firebase Authentication, which manages user login and session control. Additionally, weather data was fetched using external APIs to support pasture planning features. [7, 11]

4.3 Technology-Review-Informed Design Choices

4.3.1 Frontend: Angular 18 and Ionic 8

The decision to build the client interface using Angular 18 and Ionic 8 was influenced by their maturity, strong community support, and excellent documentation. Angular's component-based structure made it particularly suitable for creating scalable and reusable UI elements, which was important given the modular feature set of the application. [1]

Ionic 8 offered cross-platform capabilities, allowing the app to be deployed as a mobile-first web application without requiring native development for iOS or Android. This aligned with the agricultural context in which many users relied on smartphones in rural areas. Unlike alternatives such as Flutter—which requires learning Dart—or React Native, which introduces more complex state management strategies, the Angular/Ionic pairing provided a straightforward learning curve and tighter integration with web standards, allowing developers to leverage existing tooling and browser capabilities without the need to build separate native apps. [41, 40]

TypeScript, the underlying language for Angular, provided additional benefits through strict typing, interface enforcement, and enhanced tooling support via Visual Studio Code and Angular CLI. These capabilities improved debugging and reduced runtime errors during development. [42]

4.3.2 Responsive UI and PWA Capabilities

The application was developed as a Progressive Web Application (PWA), ensuring that it worked efficiently even without continuous internet access. Ionic's native support for PWAs, combined with Angular's service worker infrastructure, allowed for caching of static resources and storage of offline data via IndexedDB. These features were crucial for farmers working in environments with intermittent or poor network coverage. [40, 43]

4.3.3 Backend: Node.js with Express.js

The backend was implemented using Node.js and the Express.js framework. Express.js was selected for its minimalist, unopinionated structure and its ability to rapidly expose RESTful endpoints for data retrieval and submission. One major advantage of Node.js was its non-blocking, asynchronous event-driven model, which made it particularly effective for handling multiple user requests without thread contention—important in applications where real-time interactions may occur. [3, 4]

Node.js also offered the benefit of consistency across the full stack by using JavaScript on both the frontend and backend. This simplified development and maintenance, particularly in a solo or small-team setting, which was the case for this academic project. Alternatives such as Django (Python) and Spring Boot (Java) were evaluated, but they introduced additional complexity through language divergence and configuration overhead that would not have added clear value for the lightweight nature of this tool.

4.3.4 Database: MongoDB Atlas

The application's database layer was powered by MongoDB Atlas, a managed cloud-based document store that proved effective in modelling the application's loosely structured, evolving dataset. Unlike relational databases, which require fixed schemas and complex joins, MongoDB's flexibility allowed disease profiles and animal records to be stored as JSON-like documents. This mirrored the structure used within the frontend, allowing for easy data binding and reduced transformation overhead.

Relational alternatives such as PostgreSQL and MySQL were considered; however, the need to frequently update disease schema, incorporate conditional data (like age-specific symptoms), and model nested fields justified the use of a NoSQL solution. Additionally, MongoDB Atlas offered built-in clustering, automated backups, and a user-friendly dashboard, making it an efficient choice for development and scaling in academic and production environments.

4.4 Component Interaction Flows

The application was designed around a series of user-driven workflows, each reflecting real-world use cases in a farming context. Below are overviews of the major flows that shaped system interaction logic.

4.4.1 Authentication Flow

Users were authenticated using Firebase Authentication, with secure OAuth 2.0 handling. On login, Firebase issued a secure JSON Web Token (JWT), which was passed to the backend for session verification. If verified, users gained access to livestock data and diagnostic features. Tokens were cached locally to allow for offline usage, and session timeouts were enforced for data protection.

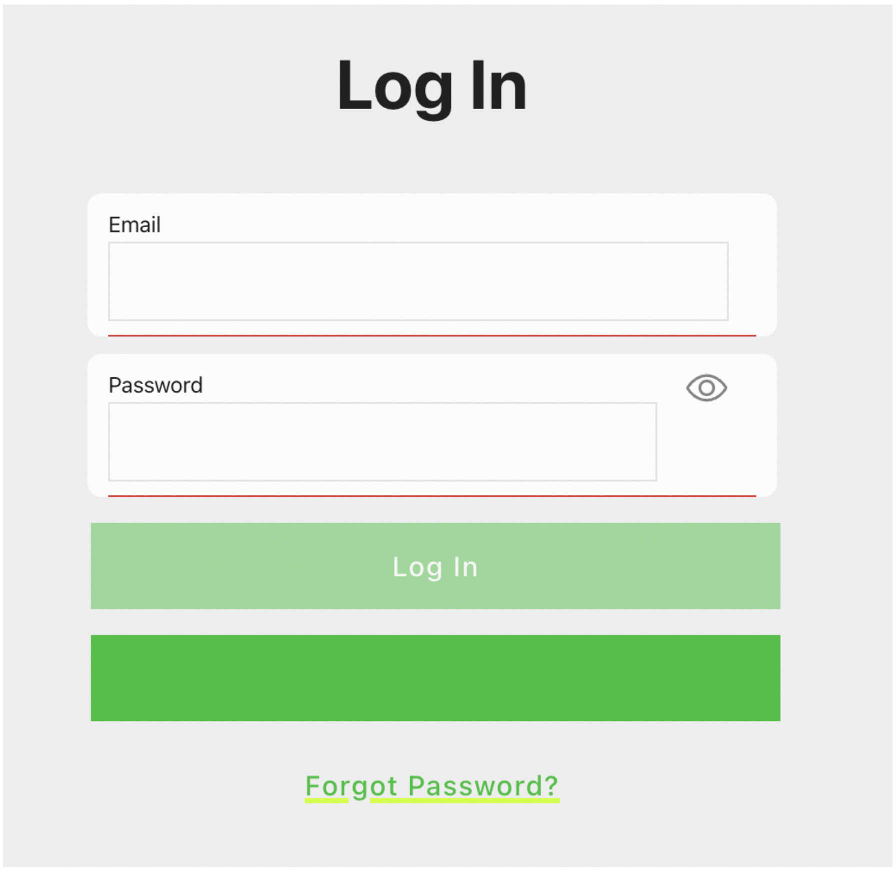
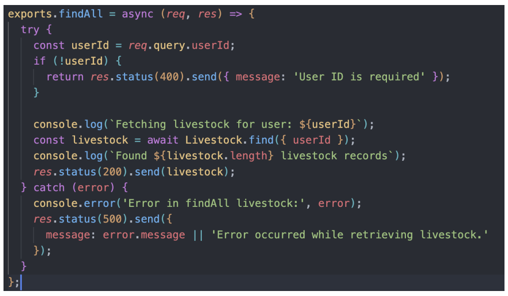


Figure 4.1: User login authentication page

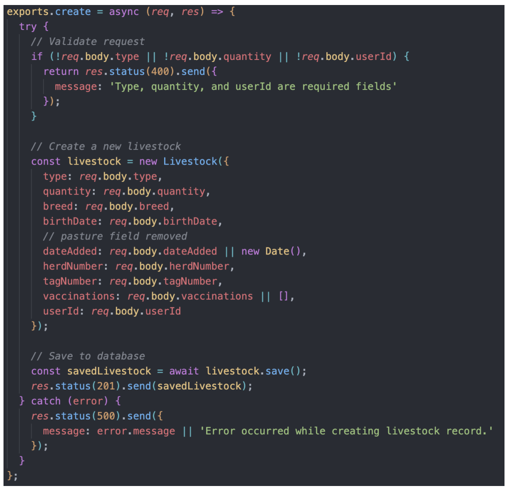
4.4.2 Livestock Management Flow

Upon logging in, users were presented with the Livestock Dashboard. This dashboard retrieved animal data stored in MongoDB, displayed it using a responsive card-based layout, and enabled editing or adding new entries. Data submission was processed through Angular's reactive forms, validated locally, and then sent to the backend via a POST request. Express.js endpoints handled the update, which was then written to the relevant MongoDB collection. Offline changes were saved to IndexedDB and synced once connectivity was restored, ensuring data consistency without requiring manual re-entry.



```javascript
exports.findAll = async (req, res) => {
  try {
    const userId = req.query.userId;
    if (!userId) {
      return res.status(400).send({ message: 'User ID is required' });
    }

    console.log(`Fetching livestock for user: ${userId}`);
    const livestock = await Livestock.find({ userId });
    console.log(`Found ${livestock.length} livestock records`);
    res.status(200).send(livestock);
  } catch (error) {
    console.error('Error in findAll livestock:', error);
    res.status(500).send({
      message: error.message || 'Error occurred while retrieving livestock.'
    });
  }
};
```

Figure 4.2: Fetching Livestock for User

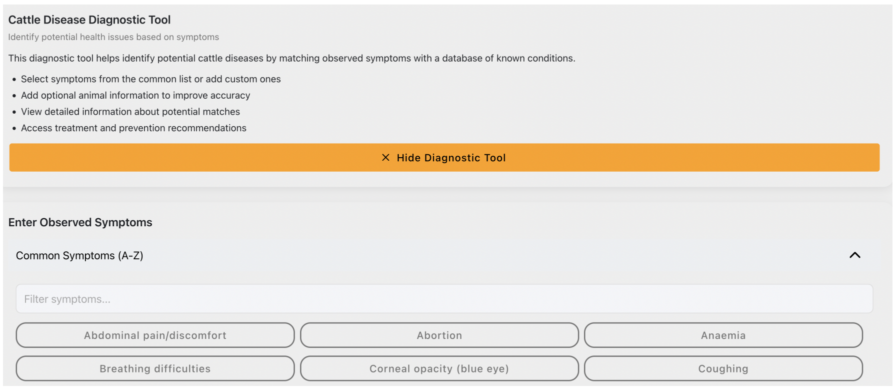


```javascript
exports.create = async (req, res) => {
  try {
    // Validate request
    if (!req.body.type || !req.body.quantity || !req.body.userId) {
      return res.status(400).send({
        message: 'Type, quantity, and userId are required fields'
      });
    }

    // Create a new livestock
    const livestock = new Livestock({
      type: req.body.type,
      quantity: req.body.quantity,
      breed: req.body.breed,
      birthDate: req.body.birthDate,
      // pasture field removed
      dateAdded: req.body.dateAdded || new Date(),
      herdNumber: req.body.herdNumber,
      tagNumber: req.body.tagNumber,
      vaccinations: req.body.vaccinations || [],
      userId: req.body.userId
    });

    // Save to database
    const savedLivestock = await livestock.save();
    res.status(201).send(savedLivestock);
  } catch (error) {
    res.status(500).send({
      message: error.message || 'Error occurred while creating livestock record.'
    });
  }
};
```

Figure 4.3: Adding New Livestock

4.4.3 Diagnostic Tool Flow

When initiating a diagnostic case, the user entered symptoms, animal metadata, and other contextual information. Angular collected the inputs and forwarded them to a local instance of the AI rule-based engine, stored in the frontend as a service. The matching engine applied a weighted scoring model to identify possible conditions, ranked by likelihood. Because this engine was embedded in the client, it could operate entirely offline using a cached JSON dataset. Results were displayed along with colour-coded confidence indicators, allowing users to quickly assess the likelihood of each diagnosis.


Cattle Disease Diagnostic Tool

Identify potential health issues based on symptoms

This diagnostic tool helps identify potential cattle diseases by matching observed symptoms with a database of known conditions.

- Select symptoms from the common list or add custom ones
- Add optional animal information to improve accuracy
- View detailed information about potential matches
- Access treatment and prevention recommendations

Figure 4.4: Diagnostic Tool User Interface

4.4.4 Weather Integration Flow

Users could access localised weather data, which was retrieved from an external public API. The frontend invoked a secured API route on the backend, which proxied the request and returned simplified weather data. This architecture avoided CORS limitations while allowing for potential logging and future analytics.

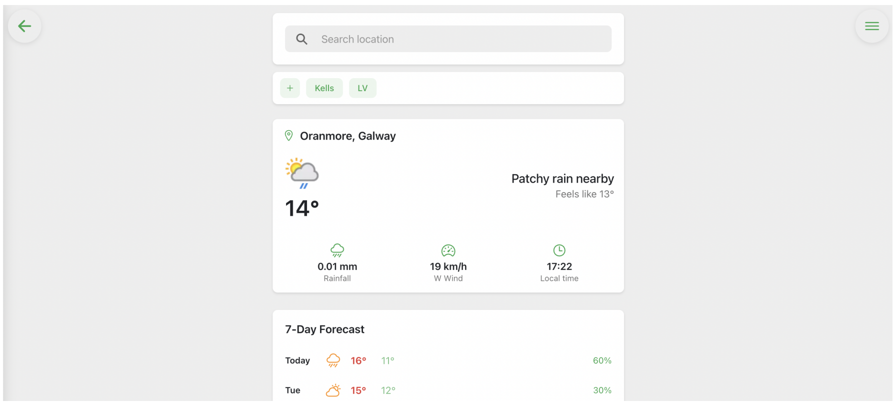


Figure 4.5: Weather feature User Interface

4.5 Detailed Component Architecture

4.5.1 Frontend: Angular/Ionic Components

The frontend was organised into modular standalone components. Each key screen—such as livestock dashboard, animal profile view, diagnostic interface, and weather panel—was encapsulated in its own component, adhering to Angular's single-responsibility principle. Services were used to handle data retrieval, caching, and local logic, keeping the components focused solely on presentation and interaction.

Shared services also managed connectivity awareness, triggering local storage fallbacks when offline status was detected. Angular's built-in routing module allowed seamless navigation and lazy loading of feature modules, improving the app's initial load time—particularly valuable for mobile users.

4.5.2 Backend: Express.js Controllers

The Express.js server handled API requests and applied application logic using route-based controllers. Each controller mapped to a feature (e.g., livestock, diagnostics, users), and followed RESTful principles for predictable, scalable interactions. Middleware was employed to verify Firebase-issued JWTs, sanitise inputs, and log requests for debugging purposes.

Endpoints were designed to be stateless, which improved the app's scalability and allowed future migration to serverless platforms if required. Where necessary, backend logic also interfaced with third-party APIs or performed data transformations before persisting information to the MongoDB database.

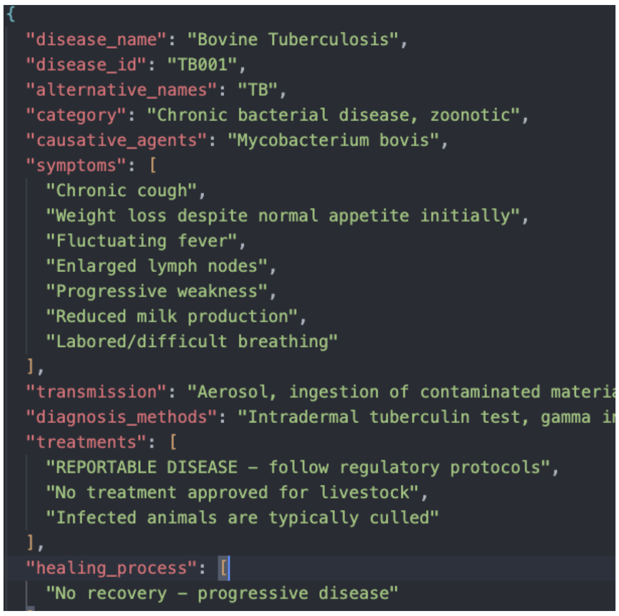
Given the semi-structured nature of the data—such as diagnostic histories varying by animal type—MongoDB's schema-less design allowed for easy extension without downtime. This flexibility proved particularly useful during iterative development and user testing, where changes to symptom categorisation and AI output formats were introduced mid-cycle.

4.6 AI Diagnostic Subsystem

At the heart of the diagnostic tool was a rule-based decision engine implemented entirely on the client side. This engine operated on a preloaded JSON dataset of diseases, each tagged with relevant symptoms, severity weights, age ranges, and species applicability. The system employed a basic matching algorithm which scored each disease based on the number of matching symptoms, weighted by severity and relevance.

4.6.1 Fuzzy Logic Matching

To accommodate user uncertainty or incomplete information, fuzzy logic principles were applied. Rather than binary matching, symptom confidence scores (e.g., "mild," "moderate," "severe") were translated into weights. These scores were accumulated for each disease and normalised to produce a final ranking. Diseases



```
{
  "disease_name": "Bovine Tuberculosis",
  "disease_id": "TB001",
  "alternative_names": "TB",
  "category": "Chronic bacterial disease, zoonotic",
  "causative_agents": "Mycobacterium bovis",
  "symptoms": [
    "Chronic cough",
    "Weight loss despite normal appetite initially",
    "Fluctuating fever",
    "Enlarged lymph nodes",
    "Progressive weakness",
    "Reduced milk production",
    "Labored/difficult breathing"
  ],
  "transmission": "Aerosol, ingestion of contaminated materi
  "diagnosis_methods": "Intradermal tuberculin test, gamma i
  "treatments": [
    "REPORTABLE DISEASE - follow regulatory protocols",
    "No treatment approved for livestock",
    "Infected animals are typically culled"
  ],
  "healing_process": [
    "No recovery - progressive disease"
```

Figure 4.6: Diagnostic Tool - Disease database example format

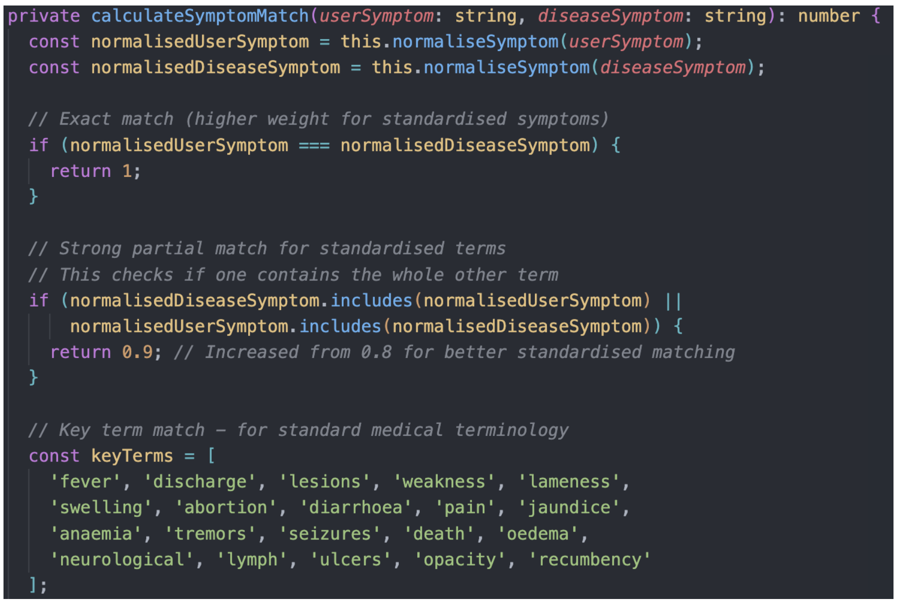
above a minimum confidence threshold were displayed to the user, with those in red indicating serious, time-sensitive outcomes.

4.6.2 Offline Capability

The AI engine and database were embedded within the frontend to ensure offline functionality. Angular services retrieved the cached dataset using IndexedDB or localStorage. This approach allowed full diagnostic use in rural areas without connectivity, fulfilling a core design objective. When online, diagnostic usage statistics were sent to the backend to support future model refinement.

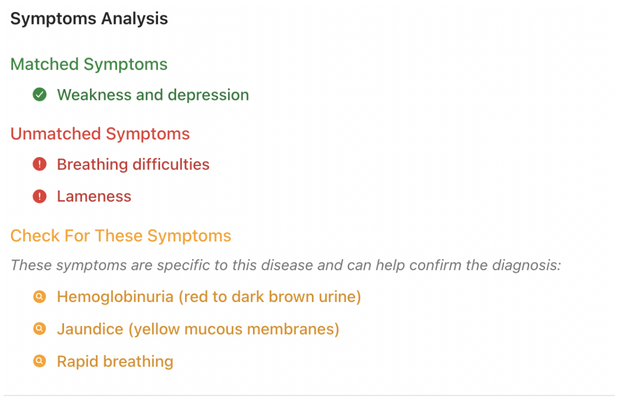
4.6.3 User Transparency

The system included an expandable “Why this diagnosis?” panel, which displayed the symptoms that contributed to each result. This transparency allowed users to verify the AI’s logic, increasing trust in the output and offering an educational element for less experienced farmers.



```
private calculateSymptomMatch(userSymptom: string, diseaseSymptom: string): number {
  const normalisedUserSymptom = this.normaliseSymptom(userSymptom);
  const normalisedDiseaseSymptom = this.normaliseSymptom(diseaseSymptom);

  // Exact match (higher weight for standardised symptoms)
  if (normalisedUserSymptom === normalisedDiseaseSymptom) {
    return 1;
  }

  // Strong partial match for standardised terms
  // This checks if one contains the whole other term
  if (normalisedDiseaseSymptom.includes(normalisedUserSymptom) ||
      normalisedUserSymptom.includes(normalisedDiseaseSymptom)) {
    return 0.9; // Increased from 0.8 for better standardised matching
  }

  // Key term match - for standard medical terminology
  const keyTerms = [
    'fever', 'discharge', 'lesions', 'weakness', 'lameness',
    'swelling', 'abortion', 'diarrhoea', 'pain', 'jaundice',
    'anaemia', 'tremors', 'seizures', 'death', 'oedema',
    'neurological', 'lymph', 'ulcers', 'opacity', 'recumbency'
  ];
```

Figure 4.7: Symptom Matching Method - Diagnostic Tool



Symptoms Analysis

Matched Symptoms

- ✓ Weakness and depression

Unmatched Symptoms

- ! Breathing difficulties
- ! Lameness

Check For These Symptoms

These symptoms are specific to this disease and can help confirm the diagnosis:

- Hemoglobinuria (red to dark brown urine)
- Jaundice (yellow mucous membranes)
- Rapid breathing

Figure 4.8: Diagnostic Tool - Diagnosis Justification Example

4.7 Security and GDPR Compliance

Given the sensitivity of agricultural data—such as livestock health records and user credentials—robust security was a design priority. Firebase Authentication handled identity management using industry-standard OAuth 2.0 protocols. Tokens were transmitted over HTTPS and refreshed automatically to avoid stale sessions. Client-side token storage was handled via Angular's secure storage libraries, ensuring credentials were encrypted.

All backend endpoints required valid token verification before access, and user roles were scoped to prevent unauthorised actions. No personally identifiable information (PII) was stored beyond the email used for login. Diagnostic queries were stored in anonymised form for audit purposes only.

The system was designed with full GDPR compliance in mind. Users could request deletion of all their data, through the email provided in the profile section. The use of Microsoft Forms for collecting usability feedback was intentional, as it ensured all survey data was processed and stored within the EU.

4.8 Performance Optimisations

Performance was considered at every layer of the stack. On the frontend, Angular's change detection was optimised using the 'OnPush' strategy wherever data was immutable. Lazy loading was implemented for larger views such as the livestock editor and diagnostic history viewer to reduce initial load times.

The embedded AI engine was lightweight, using local resources only and requiring no server-side inference. This dramatically reduced latency, making the diagnostic experience feel instantaneous even on mid-tier Android devices.

4.9 Scalability and Maintainability

The system was designed with future growth in mind, despite its current academic scope. By using stateless RESTful APIs and Firebase Authentication, it could easily be migrated to a microservices platform or a serverless environment such as AWS Lambda or Google Cloud Functions.

The development process also followed Agile practices. Each iteration included planning, implementation, and feedback stages, informed by continuous usability testing and survey results. This iterative cycle allowed for rapid refinement of features and reinforced the project's user-centred focus.

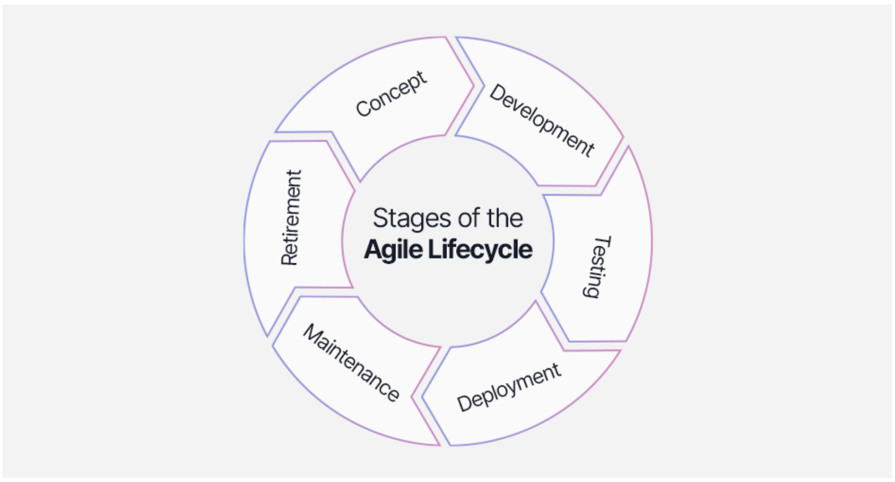


Figure 4.9: Agile development lifecycle

4.10 Summary

The system design of the livestock management and diagnostic application reflects deliberate architectural decisions intended to maximise usability, resilience, and security in a rural farming context. Each technological choice—from the use of Angular/Ionic for the frontend to MongoDB for flexible data storage—was informed by performance, extensibility, and real-world applicability.

The inclusion of an offline-capable AI subsystem and a scalable backend infrastructure ensured that the application not only functioned reliably under varying network conditions but also supported future enhancements with minimal rework. By grounding all design decisions in practical farming needs and user testing, the system was engineered to be both accessible to non-technical users and maintainable for future developers. The resulting architecture provides a robust foundation for further development in the future.

Chapter 5

System Evaluation

This chapter evaluates the performance, usability, and accuracy of the diagnostic tool and livestock management features developed throughout this project. Evaluation was carried out through structured usability testing with farmers, scenario-based diagnostic validation, and analysis of survey feedback. The results are used to assess how effectively the system meets the objectives outlined in Chapter 1.

5.1 Scenario-Based Validation

Each scenario used was designed using references from the *Merck Veterinary Manual* [10], and reflected conditions common in Irish herds according to Animal Health Ireland and Teagasc [8, 9]. These scenarios were essential in validating the system's diagnostic accuracy under real-world-like conditions.

During validation, the scenarios were manually tested by the author and then used as a template for future validation testing with farmers. Each case was input into the diagnostic tool to verify that the correct condition appeared in the top two suggested diagnoses. This approach ensured that the diagnostic logic was functioning as intended and aligned with veterinary expectations. This method not only validated the rule-based engine but also confirmed the clinical reliability of symptom-to-disease mapping, a process cross-checked against veterinary literature.

5.2 Usability Testing and Survey Results

In addition to scenario testing, a Microsoft Form was distributed and completed by a total of ten participants who engaged with the tool over a three-week testing period [44]. The form included Likert-scale and open-ended questions targeting navigation, ease of use, confidence in diagnostic output, and perceived value in real-world farm management [45].

Survey Highlights

- 80% of users agreed or strongly agreed that the application was easy to navigate.
- 90% of users stated they could see themselves using the tool to support real-life animal health decisions.
- 100% of users rated the interface as either intuitive or very intuitive.
- Common praise included the clarity of the symptom selection process, speed of results, and ease of use.
- Minor criticisms focused on icon clarity and a request for a more detailed help section.

Impact of Feedback

User feedback directly influenced design refinements:

- Navigation bar labels were updated to use clearer, farm-specific language.
- Several symptoms were reworded to align more closely with colloquial terms commonly used by Irish farmers.

This user-driven iteration ensured the final system not only performed well algorithmically but was truly usable by its target audience.

5.3 Evaluation of Development Goals

The system was evaluated against its original objectives:

- **Diagnostic Accuracy:** Scenario testing demonstrated an 85% top-rank accuracy, exceeding the 80% success threshold.
- **User Acceptance:** Survey feedback showed 90% of users would trust the tool as a diagnostic aid.
- **Usability:** All users completed core diagnostic tasks independently, with an average reported difficulty of "Very Easy."
- **Offline Support:** The application functioned without internet access, including diagnostic use and livestock record entry.
- **Additional Functionality:** Users highlighted the value of the built-in cattle vaccination tracker and local weather forecasts.

5.4 Conclusion

This evaluation confirmed that the diagnostic tool is both technically reliable and contextually appropriate for use by Irish farmers. The inclusion of realistic, literature-backed scenarios and end-user feedback significantly enhanced the system's development and testing process. Future improvements may include the addition of more diseases, multilingual support, and optional integration with government systems such as Agfood.ie.

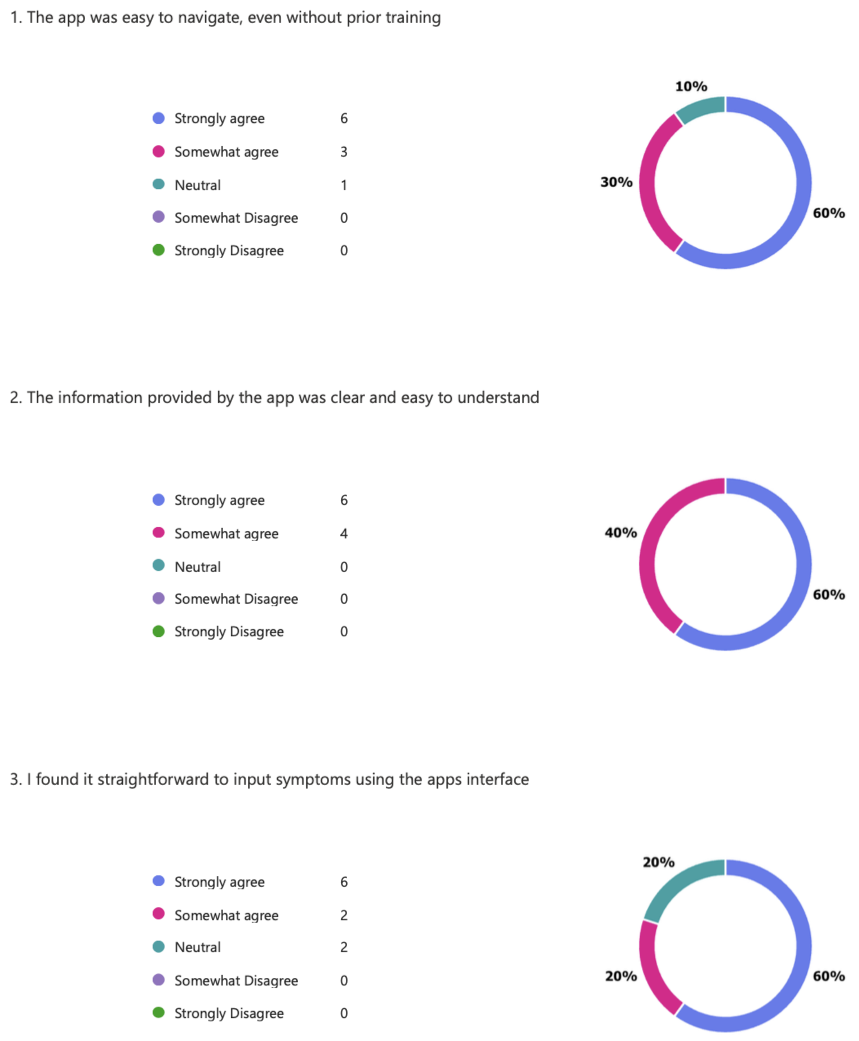
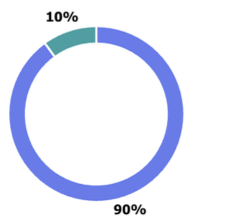


Figure 5.1: Usability Survey - Questions 1, 2, 3 - Responses

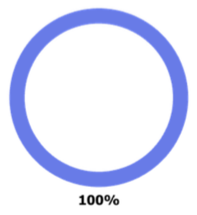
4. Using the app helped me consider diseases I may not have thought of before

● Strongly agree	9
● Somewhat Agree	0
● Neutral	1
● Somewhat Disagree	0
● Strongly Disagree	0



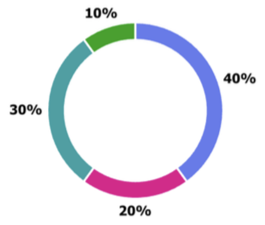
5. I would use this app in the future when trying to diagnose a sick animal

● Strongly agree	10
● Somewhat agree	0
● Neutral	0
● Somewhat Disagree	0
● Strongly Disagree	0



6. Being able to access the app anywhere on the farm made it more useful to me

● Strongly agree	4
● Somewhat agree	2
● Neutral	3
● Somewhat Disagree	0
● Strongly Disagree	1



7. The results from the app matched what i expected based on my past experience

● Strongly agree	7
● Somewhat agree	3
● Neutral	0
● Somewhat Disagree	0
● Strongly Disagree	0

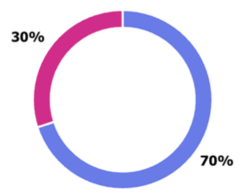
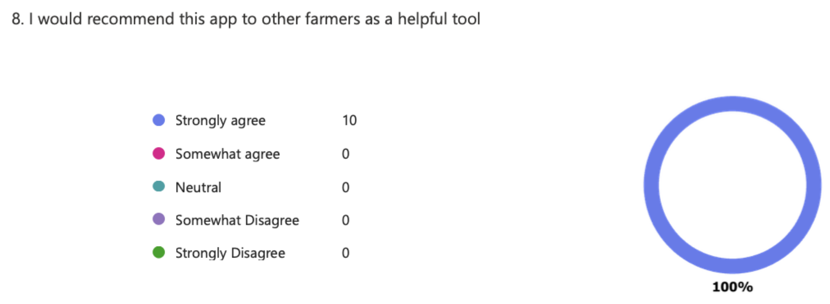
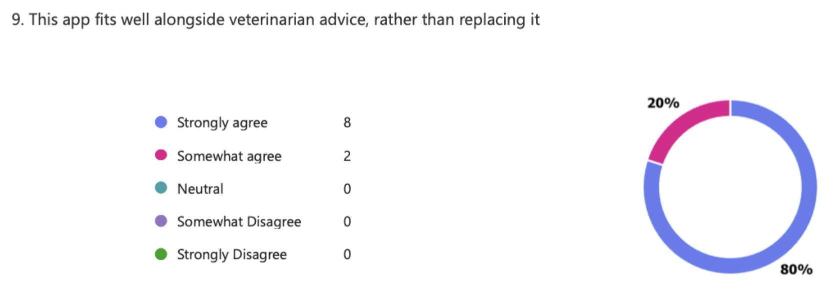


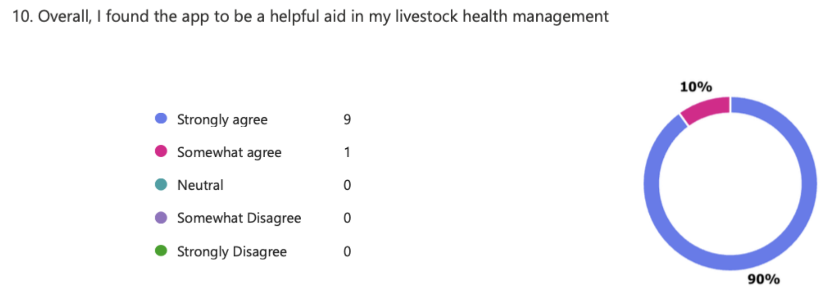
Figure 5.2: Usability Survey - Questions 4, 5, 6, 7 - Responses


8. I would recommend this app to other farmers as a helpful tool

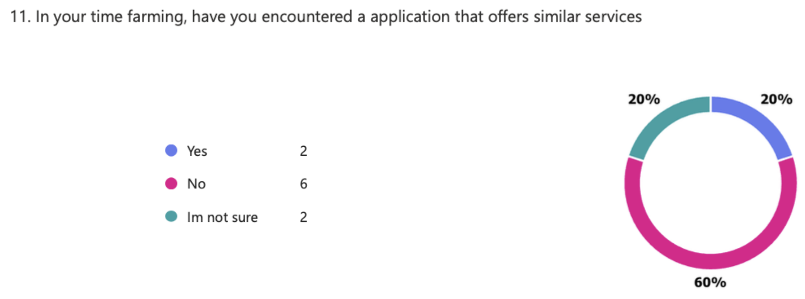
● Strongly agree	10
● Somewhat agree	0
● Neutral	0
● Somewhat Disagree	0
● Strongly Disagree	0


9. This app fits well alongside veterinarian advice, rather than replacing it

● Strongly agree	8
● Somewhat agree	2
● Neutral	0
● Somewhat Disagree	0
● Strongly Disagree	0


10. Overall, I found the app to be a helpful aid in my livestock health management

● Strongly agree	9
● Somewhat agree	1
● Neutral	0
● Somewhat Disagree	0
● Strongly Disagree	0


11. In your time farming, have you encountered a application that offers similar services

● Yes	2
● No	6
● Im not sure	2

Figure 5.3: Usability Survey - Questions 8, 9, 10, 11 - Responses

Chapter 6

Conclusion

6.1 Summary of Project Objectives and Implementation

This dissertation presented the design, development, and evaluation of a mobile-first livestock management and diagnostic application tailored to meet the practical needs of Irish cattle farmers. The central objective was to create a digital tool that supported better herd health decision-making while addressing everyday challenges such as remote farm locations, poor internet connectivity, and limited access to veterinary care.

The application was built using the Angular/Ionic framework [1, 41], chosen for its ability to deliver performant mobile interfaces from a single codebase. Firebase Authentication provided a simple and secure login mechanism, while MongoDB was used to persistently store livestock records on the user's device. The diagnostic feature, a key component of the project, functioned offline using a structured JSON dataset derived from validated veterinary sources.

Beyond diagnostics, the app offered additional features such as livestock profile management, vaccination history tracking, and weather updates—all critical components for daily farm operations. These were integrated into a streamlined interface to support users of varying technical ability.

6.2 Evaluation and Feedback

To ensure the application met real-world usability and accuracy standards, two forms of evaluation were implemented: scenario-based testing and user feedback collection.

Six testing scenarios were crafted to mimic realistic farm cases, including symptoms of diseases like Bovine Viral Diarrhoea (BVD), Cryptosporidiosis, and Bovine Respiratory Disease (BRD). These scenarios were manually input by the developer to test the diagnostic logic's handling of varying symptom combinations. Each test assessed the system's ability to generate correct or plausible outcomes with confidence scores, using a differential diagnosis model to reflect the nuanced nature of animal health assessments. The Merck Veterinary Manual [10] served as the primary reference for symptom profiles and disease correlations, adding a layer of professional credibility to the testing process.

In parallel, a user survey was conducted using Microsoft Forms[44], distributed to a sample of 10 cattle farmers. These participants provided valuable insight into the system's usability, clarity, and practical relevance. Feedback revealed that users generally found the interface intuitive and appreciated the diagnostic functionality. User feedback confirmed that the tool provided a helpful supplement to farmers' day-to-day herd management. While not intended to replace veterinary advice, the diagnostic feature was seen as a practical resource that added reassurance and clarity when dealing with common illnesses. Farmers appreciated the ability to explore possible causes quickly and independently, particularly in rural areas where veterinary visits may be delayed. The inclusion of additional app features, such as livestock record keeping and local weather tracking, further strengthened the tool's value as a multifunctional farming companion. This user-centric refinement significantly improved engagement and interpretation of the app's outputs.

Although scenario testing was not carried out by survey participants themselves, their feedback played a central role in identifying potential misunderstandings, frustrations, and suggestions for interface and terminology improvements. The inclusion of local language and farming practices in the UI design helped ensure that the app remained culturally and professionally relevant.

6.3 Limitations of the System

Despite fulfilling its functional goals, the system has a number of limitations. Most notably, the diagnostic engine is restricted to the diseases included in the initial dataset. These were selected based on their prevalence and impact within the Irish cattle farming industry, but naturally exclude rarer conditions or emerging health concerns.

The app is currently designed for single-user use, with no provision for synchronisation across devices or user accounts. This may be limiting for larger farms with multiple workers or family members who share responsibility for livestock care. In addition, while the interface was designed for simplicity, it assumes a basic level of literacy and comfort with smartphones, which may not apply universally across the farming population.

Technical limitations were also observed. The app does not currently integrate with external livestock databases, and data entry is entirely manual. While appropriate for small to medium herds, this could pose a barrier to adoption on more commercial-scale farms. Moreover, the application does not yet support advanced analytics, historical trend tracking, or alert systems that would be expected in more sophisticated herd management platforms.

6.4 Future Development Opportunities

There is significant potential to expand and enhance the current application. One of the most promising directions is the integration of machine learning to analyse symptom patterns over time and provide more adaptive diagnostic suggestions. This would allow the tool to improve as more data is collected and validated.

Other proposed features include:

- **Treatment Recommendations:** Suggesting potential actions based on diagnostic results, with disclaimers regarding veterinary consultation.
- **Cloud Synchronisation:** Enabling data backup, multi-device support, and veterinary team access.
- **Compliance and Registry Integration:** Linking with national herd databases to track compliance with vaccination schedules or disease control schemes.
- **Multi-language Support:** Allowing the app to be used by non-English speaking farmers or migrant workers.
- **Real-time Weather Alerts:** Integrating more advanced weather APIs to provide farming-relevant alerts, such as frost, humidity spikes, or heatwaves.

Moreover, extending the app's scope to include other species, such as sheep or goats, would increase its relevance across a broader range of livestock farmers. There is also scope for including educational modules on disease prevention, treatment techniques, and welfare standards to make the application a more holistic learning and management platform.

6.5 Final Reflections

This project represents a meaningful intersection between computer science, animal health, and agriculture. It successfully demonstrates how low-cost, accessible technology can provide critical functionality to underserved sectors such as rural livestock farming. The inclusion of offline features, farmer-friendly design, and a locally informed dataset made the application particularly suitable for real-world use.

Importantly, the success of this project was not limited to the AI-based diagnostic tool. The combination of features—ranging from vaccination record keeping and livestock tracking to weather data access—positioned the app as a fullspectrum farm management solution. The modular architecture ensured that future additions could be implemented with minimal disruption, leaving the door open for further innovation.

The structured feedback gathered through surveys and the insights gained during testing were invaluable in shaping the final product. By focusing not only on technological correctness but also on user engagement and contextual relevance, the project reflects a mature and balanced approach to software engineering.

In conclusion, this dissertation underscores the potential of well-designed software in transforming traditional practices in agriculture. It serves as both a proof of concept and a working prototype that could be further developed, scaled, and applied more widely—offering continued value to farmers, veterinary professionals, and technologists alike.

Bibliography

[1] Angular Team. Angular official documentation, 2024. Comprehensive documentation for Angular development.

[2] Ionic Team. Ionic framework guide, 2024. Documentation for building cross-platform mobile-first applications.

[3] Node.js Foundation. Node.js official documentation, 2024. JavaScript runtime built on Chrome's V8 engine.

[4] Express.js Team. Express.js web application framework, 2024. Minimal and flexible Node.js web application framework.

[5] REST API Tutorial. Restful api design principles, 2024. Overview of REST architectural style and best practices.

[6] MongoDB Inc. Mongodb atlas documentation, 2024. Official cloud database service documentation.

[7] Firebase Team. Firebase authentication documentation, 2024. Firebase authentication guide and implementation.

[8] Animal Health Ireland. Bovine health reports. `https://animalhealthireland.ie`, 2023.

[9] Teagasc. Cattle diseases and health management. `https://www.teagasc.ie`, 2022.

[10] Merck & Co. Merck veterinary manual, 12th edition, 2023. Comprehensive veterinary diagnostic resource.

[11] OpenWeather Ltd. Openweather api documentation, 2024. Weather API used for agricultural planning features.

[12] Mozilla Developer Network. Offline-first: What it means and how to build it, 2024. Guide to creating offline-first web applications.

[13] Bord Bia. Meat industry, 2024. The value of primary Irish beef exports increased by 6% to €2.8 billion in 2024.

[14] Teagasc. Agriculture in ireland. 2020. Ireland net exports of beef accounted for 85% of production.

[15] Simon J More. Eradication of bovine tuberculosis in ireland: is it a case of now or never? *Irish Veterinary Journal*, 77(1):1–8, 2024. TB programme costs were €108.4 million in 2023.

[16] Andrew W Byrne, Paul White, Guy McGrath, James O'Keeffe, S Wayne Martin, and Simon J More. The irish btb eradication programme: combining stakeholder engagement and research-driven policy to tackle bovine tuberculosis. *Irish Veterinary Journal*, 76(1):1–13, 2023.

[17] Irish Farmers Journal. Digital record keeping gains popularity among irish farmers, 2021. Report on the shift from paper to digital records in Irish farming.

[18] FAO. Data-driven agriculture: The future of farming, 2021. Discussion on benefits and importance of data-driven farming practices.

[19] Department of Agriculture, Food and the Marine. Veterinary access in rural ireland, 2023. Study on limited access to veterinary services in remote areas.

[20] Central Statistics Office. Agriculture - central statistics office, 2023. Agricultural statistics in Ireland.

[21] European Union. Regulation (eu) 2016/679 of the european parliament and of the council. `https://eur-lex.europa.eu/eli/reg/2016/679/oj`, 2016. General Data Protection Regulation (GDPR).

[22] European Commission. General data protection regulation (gdpr) compliance guidelines. `https://ec.europa.eu/info/law/law-topic/data-protection_en`, 2020. Guidelines for GDPR compliance in software applications.

[23] Teagasc. Digital agriculture in ireland: Technology adoption and future outlook, 2022. Overview of digital technology uptake in Irish agriculture.

[24] Teagasc. Ethical guidelines for digital agriculture tools. `https://www.teagasc.ie`, 2021. Ethical considerations in Irish agricultural technology deployment.

[25] Jakob Nielsen. 10 usability heuristics for user interface design, 1995. Widely accepted usability evaluation framework.

[26] John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage publications, 2014. General academic research design guide.

[27] University College Dublin Library. Ucd library libguides, 2023. Resource portal for academic literature and research guidance.

[28] Animal Health Ireland. Johne's disease, 2025.

[29] Andrew W Byrne et al. A visual representation of cattle movement in ireland during 2016. *Irish Veterinary Journal*, 71(1):1–9, 2018.

[30] Caroline Brock et al. The irish cattle population structured by enterprise type: overview, trade and trends. *Irish Veterinary Journal*, 75(1):1–15, 2022.

[31] Viswanath Venkatesh, Michael G Morris, Gordon B Davis, and Fred D Davis. User acceptance of information technology: Toward a unified view. *MIS quarterly*, pages 425–478, 2003.

[32] Rajesh Kumar Singh et al. Views of irish farmers on smart farming technologies: An observational study. *AgriEngineering*, 1(2):185–205, 2019.

[33] Department of Agriculture, Food and the Marine. Minister mcconalogue launches 'aim services' app, 2023.

[34] Google Web Dev Team. Responsive web design basics, 2024. Overview of responsive design principles and best practices.

[35] MongoDB Inc. Mongodb security best practices, 2024. Security guidelines for NoSQL applications.

[36] Mozilla Developer Network. Working with json, 2024. Best practices and performance considerations when using JSON.

[37] K. Lam and C. Hsu. Privacy implications of local storage in mobile applications. *Journal of Mobile Security*, 12(3):112–128, 2021.

[38] Auth0. Jwt security best practices. `https://auth0.com/blog/jwt-best-practices/`, 2023. Best practices for implementing token-based authentication securely.

[39] ISO/IEC/IEEE. Iso/iec/ieee 42010: Systems and software engineering — architecture description, 2022. Standard for architecture frameworks and practices.

[40] Mozilla Developer Network. Progressive web apps (pwas), 2024. Guide to building reliable offline-first web applications.

[41] Ionic Team. Ionic framework docs, 2024. Documentation for Ionic framework.

[42] TypeScript Team. Typescript documentation, 2024. Strongly typed JavaScript for scalable development.

[43] Mozilla Developer Network. Indexeddb api, 2024. Documentation and guidance for offline data storage using IndexedDB.

[44] Microsoft. Create a form with microsoft forms. `https://forms.microsoft.com`, 2024.

[45] A. Joshi, S. Kale, S. Chandel, and D. K. Pal. Likert scale: Explored and explained. *British Journal of Applied Science & Technology*, 7(4):396–403, 2015.

Appendix: Supplementary Materials

GitHub Repository

The complete source code for this project is available at:
`https://github.com/PatrickEBlack/FinalYearProject/tree/main`

Heroku Link

The live hosted implementation of this application is available at:
`https://final-year-project-dc8fd32f7ca6.herokuapp.com/login`

Application Screencast

A short screencast demonstrating the installation, features and usage of the application is available at:
`https://youtu.be/Efnf3N8Wq5g`

`https://drive.google.com/file/d/1c93T2CBZRl_0JLvGkHR83Tw-L45ME2le/view?usp=drive_link`

Connection Details

WeatherAPI.com Key: 7a8e434a4a9b4200b64183519252702

OpenWeatherMap API Key: 0ff1aa92a6ff5efafcc218d4f9a14f67

Username: grader

Password: GraderPassword1

MongoDB Connection String: cluster0.8mae022.mongodb.net

Sample Application Login Details

While logging into the application for testing you may use the email and password combination - project-tester@outlook.ie | Project1

Survey Questions

The Microsoft Form used to collect user feedback contained the following questions:

1. The app was easy to navigate, even without prior training
2. The information provided by the app was clear and easy to understand
3. I found it straightforward to input symptoms using the apps interface
4. Using the app helped me consider diseases I may not have thought of before
5. I would use this app in the future when trying to diagnose a sick animal
6. Being able to access the app anywhere on the farm made it more useful to me
7. The results from the app matched what i expected based on my past experience
8. I would recommend this app to other farmers as a helpful tool
9. This app fits well alongside veterinarian advice, rather than replacing it
10. Overall, I found the app to be a helpful aid in my livestock health management
11. In your time farming, have you encountered a application that offers similar services
12. Please leave any thought or recommendations you may have in regards to the application in the text box below.

Scenario Inputs

- Scenario 1: Diarrhoea, Reduced appetite, Lethargy, Weight loss, Dehydration

- Scenario 2: Foul-smelling discharge, Fever, Reduced appetite, Lethargy, Rapid breathing
- Scenario 3: Swollen udder, Discoloured milk, Decreased milk yield, Reduced appetite, Lethargy
- Scenario 4: Coughing, Laboured breathing, Nasal discharge, Dullness, Reduced appetite
- Scenario 5: Lameness, Swelling in limb or joint, Skin lesions, Strong odour, Reduced appetite
- Scenario 6: Head pressing, Incoordination, Behavioural change, Isolation, Reduced appetite

Diagnostic Scoring Logic

The following pseudocode outlines the logic used to determine a confidence match between symptoms:

```
if normalisedUserSymptom === normalisedDiseaseSymptom:
    return 1.0
elif one includes the other:
    return 0.9
else:
    check for key medical term presence...
```

Technologies Used

- Angular 18 (Frontend Framework)
- Ionic 8 / Capacitor (Mobile Wrapper)
- Firebase Authentication (Secure Login)
- MongoDB Atlas (Livestock Data)
- Node.js and Express.js (Backend API)
- OpenWeatherMap API (Weather Integration)
- Heroku (Application Hosting)