

Bachelor of Information Technology

PRJ703



Modernising a Legacy Food Production Master Data Management System: From Visual Basic to Modern Technologies

Jayden Houghton

NMIT Student ID: 13524050

Supervisor: Todd Cochrane

Bachelor of Information Technology

PRJ703 - 2025

Acknowledgements

I would like to express my appreciation to the Talley's Group, especially to my manager, for taking me on and allowing me to do my project with the company. I would also like to express my gratitude to all the developers and other team members at Talley's who helped me throughout the project. You were always willing to answer my questions and help me solve problems even though you had full workloads yourself. Working within the development team has offered me valuable knowledge and experience that has helped me grow both technical and non-technical skills.

Lastly, a thank you to all my tutors who have taught me so much and supported me throughout this education journey.

Abstract

This report discusses the modernisation project undertaken to migrate Talley's legacy master data management system from Visual Basic 6 into a React web client with a C# .NET and Entity Framework Core API. With Visual Basic development having lost support from Microsoft, development of the previous system was becoming more difficult and outdated, raising issues around security, performance, and maintainability. This led Talley's to approve the modernisation of the system and migration to modern technologies.

Developing the web client involved replicating the existing user interfaces from the Visual Basic application in the web client, as well as implementing the frontend logic required to interact with the API and display data correctly. Expanding the API involved adding the required models, entity schema, mapping profiles, controllers, and services to enable the implemented functionality, as well as writing expansive xUnit tests for the API. The existing Microsoft SQL Server database also required minor improvements to allow for correct handling of data by Entity Framework Core.

This report explains how the features were implemented across the technologies stack, as well as a brief analysis of the technologies and architectural patterns used in the system. The project did encounter some challenges, primarily vague feature requirements requiring more iterative feedback cycles to achieve a satisfactory end product, as well as undefined migration requirements for the new system, giving poor direction for what needed to be included. Low priority work in comparison to the other developer's also led to longer feedback cycles and more repeated issues, causing increased work and a less efficient development process.

By the end of the project, 70% of the total estimated sections were migrated, which is at the upper threshold of the originally estimated 50-70% completion. This project has provided Talley's with useful modern system to replace their legacy system, that they can continue to expand upon and improve over time. The project also provided valuable experience in a real-world workplace environment, developing skills and knowledge through applied learning and exposure to new tools and processes.

Table of Contents

LIST OF FIGURES	VI
LIST OF TABLES	VI
LIST OF ABBREVIATIONS	VII
1. INTRODUCTION	1
1.1 COMPANY OVERVIEW.....	1
1.2 CURRENT SYSTEM	1
1.3 PROJECT PLAN	2
1.4 IT TEAM & MY ROLE.....	3
2 BACKGROUND.....	4
2.1 LEGACY SYSTEMS.....	4
2.2 MASTER DATA MANAGEMENT	5
2.3 MIGRATION & MODERNISATION	5
3 PLACEMENT REPORT	7
3.1 TECHNOLOGIES.....	7
3.1.1 <i>React Client</i>	7
3.1.2 <i>C# .NET API</i>	9
3.1.3 <i>SQL Server Database</i>	12
3.2 METHODOLOGY, PIPELINE & PROCESSES	13
3.2.1 <i>DevOps Culture</i>	14
3.2.2 <i>Talley's Methodology</i>	14
3.2.3 <i>Development Pipeline & Tools</i>	15
3.2.4 <i>Research Process</i>	17
3.3 CLIENT STRUCTURE	17
3.4 COMPLETED WORK	20
3.4.1 <i>CRUD Functionality</i>	22
3.4.2 <i>Special Features</i>	28
3.5 ARCHITECTURAL PATTERNS.....	35
3.5.1 <i>React Client</i>	35
3.5.2 <i>C# .NET API</i>	36
3.5.3 <i>API Tests</i>	36
3.6 VISUAL BASIC 6.0 IN 2025	36
3.7 AI IN DEVELOPMENT	37
4 ANALYSIS.....	38
4.1 SYSTEM VALUE	38
4.2 PLAN ADHERENCE.....	38
4.3 DEVELOPMENT PROCESS	40
4.4 CHALLENGES.....	40
4.5 ALTERNATIVE APPROACHES	42
4.6 INSIGHTS	43
5 FUTURE DEVELOPMENT	45
5.1 SHORT-TERM	45
5.2 MID-TERM.....	45
5.3 LONG-TERM	46
6 CONCLUSION.....	47
7 REFERENCES.....	49
APPENDIX A: AI PROMPTS FOR DEVELOPMENT	56
PROMPT 1: CUSTOM MATERIAL UI COMPONENT.....	56
PROMPT 2: VB6 FUNCTION EXPLANATION	56

List of Figures

Figure 1: Fishing methods editing page toolbar with a selection of customised MUI buttons.	8
Figure 2: Toastify error message from a failed API request.	9
Figure 3: Toastify success message from a successful API request.	9
Figure 4: Example of an xUnit fact test from Trace.	12
Figure 5: Example of an xUnit theory test from Trace.	12
Figure 6: DevOps life cycle.	13
Figure 7: DevOp's online ticket board showing in progress tickets for this project.	16
Figure 8: Structure of sections, pages, and forms in the client application.	18
Figure 9: Trace species list page.	18
Figure 10: Trace species editing page with editing form.	19
Figure 11: Trace product fishing methods editing page with editing form.	19
Figure 12: Burndown chart demonstrating the work completed over the project's duration. ...	20
Figure 13: Example of data retrieval in the React client from a two-page section.	23
Figure 14: Example of data retrieval in the React client from a single-page section.	23
Figure 15: Example of requesting and processing editing data.	24
Figure 16: Example of an item data context interface.	24
Figure 17: Example of getting editing data in an editing form.	24
Figure 18: Example of a create or update API call including client validation.	25
Figure 19: Example of two-page section editing (parent) page in requesting item data.	26
Figure 20: Example of a CreateOrUpdateItem API method using EF Core.	27
Figure 21: Example of an API delete method using EF Core.	27
Figure 22: Sync fishing methods API method.	28
Figure 23: Custom React read-only text input component.	30
Figure 24: Get skipper images API method.	31
Figure 25: Alphanumeric comparer API helper class.	32
Figure 26: Location barcode printing UI panel.	33
Figure 27: Location batch mode UI panel.	33
Figure 28: Location label printing API method.	34
Figure 29: File structure of Trace's core component.	35

List of Tables

Table 1: List of sections migrated during the project and their estimated story point value.	21
Table 2: Timeline of completed hours during the summer internship portion of the project.	39
Table 3: Scheduled vs actual timeline for the academic portion of the project.	39

List of Abbreviations

AAA	Arrange Act Assert
AI	Artificial Intelligence
API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Delivery
CRUD	Create, Read, Update, and Delete
DOM	Document Object Model
RDBMS	Relational Database Management System
REST	Representational State Transfer
EF	Entity Framework
HTML	Hypertext Markup Language
ID	Identification
IDE	Integrated Development Environment
IT	Information Technology
JSX	JavaScript XML
LINQ	Language Integrated Query
LLM	Large Language Model
MUI	Material UI
MVC	Model View Controller
OOP	Object-Orientated Programming
ORM	Object-Relational Mapper
PR	Pull Request
RC	Release Candidate
RMS	RACE Management System
SQL	Structured Query Language
UI	User Interface
VB6	Visual Basic 6
WinForms	Windows Forms
XML	Extensible Markup Language

1. Introduction

The ability to administer business data efficiently and securely is crucial for any company's success (Oracle, 2021). Business data is typically managed and communicated through legacy information systems that form the backbone of information flow in the organisation (Bisbal et al., 1997). These systems often slowly fall behind the current technologies as new software and hardware solutions rapidly evolve, restricted by their vitality to the business operations (Förnweiger, 2017). While operating as integral central hubs for the consolidation and management of vital business data (Bisbal et al., 1997), these systems are often simultaneously restricting the company from efficient and effective digital evolution (Fanelli et al., 2016; Ogunwole et al., 2023; Wolfart et al., 2021). The combination of obsolete technologies, loss of employee's expertise, perpetually increasing complexity, and poor architecture result in hard to maintain, inflexible, and costly systems that hold back businesses from maximising their innovation and adaptability to gain competitive advantage (Bisbal et al., 1997; Khadka et al., 2014; Wolfart et al., 2021).

Many businesses inevitably require major reworks to their supporting information systems (Bisbal et al., 1997). Talley's Group Ltd. (Talley's), a national food production company is no exception, having decided to invest in the modernisation of their legacy master data management system. This report will discuss the modernisation project of this system to contemporary technologies.

1.1 Company Overview

Talley's Group Ltd. (Talley's) is a New Zealand owned food production company that provides seafood, vegetables, and dairy products to both national and to international customers (Talley's, 2025-a; Talley's, 2025-b). Talley's operates nationwide, with their head office situated in Motueka and their deep-sea fleet operating out of Port Nelson (Talley's, 2025-c). Originally starting as a small local fishing business over eighty years ago, the company is now one of New Zealand's largest food production and supply groups (Talley's, 2025-b).

1.2 Current System

Talley's is currently operating using an existing legacy food processing master data management system. This program is called RACE Management System Admin (referred to as RMS Admin), as this system is used to administer the 'core' data used by the RACE Suite. Staff use RMS Admin internally to manage and administer business data such as vessel details, fish species, product details, processing plant details, storage locations, vessel owner details, and product label data. This system is crucial for ensuring that data is accurate, and that products are processed, documented, and labelled correctly.

The existing system was in development no later than 2001 (exact start date unknown) using Visual Basic 6 (VB6). Visual Basic received its last update (version 6) in 1998 (Abto Software, 2025), and while it still runs on the latest versions of Windows, Microsoft stopped providing

support for VB6 development in 2008 (Microsoft, 2024d). This means there is no official method for installing the Visual Basic Integrated Development Environment (IDE), requiring a workaround to complete the installation (Microsoft, 2024d). Microsoft's own website even states that "[they] strongly recommend that you replace your application with modern technology" (Microsoft, 2024d).

1.3 Project Plan

Since the legacy system environment has lost support many years ago, Talley's has decided to upgrade the system to modern technologies. This is likely to take advantage of recent technologies that have improved performance, scalability, and security, as well as providing a more easily maintainable system with support.

The new system, named Trace, will fill the same purpose as the legacy system, providing similar functionality. There will be minor changes, such as removing unused pages, features, and item fields, and a modernised user interface, however the primary focus is on upgrading the system to contemporary technology.

Using the legacy system as a guide, the project will be to recreate the application in the new system's environment: a React web-based client (React, n.d.a) and a C# .NET API layer (Microsoft, 2025a), working on the existing SQL Server database (Microsoft, 2025b). The migration had already been started before this project began, with the core technologies already decided, providing a strong framework to work off.

The web client is constructed using the React library as a 'framework' and utilising TypeScript (Microsoft, 2025c) to create a custom interface. The client also uses the Material UI component library (Material UI, 2025), among other popular JavaScript libraries and packages to create a clean and modern user interface (UI). This project will involve adding new pages and forms to the web client, including validation and API interfacing.

The API layer is built using C# .NET, using Entity Framework Core (EF Core) as the object relational mapper to interface with the database (Microsoft, 2024b). This layer also uses packages including AutoMapper to manage creating and mapping the database and business models (Bogard, n.d.). This project will involve the creation and expansion of controllers and services to expand the capability of the API to facilitate new client pages, as well as creating the necessary models, mapping profiles, and entity creation schemas. Integration and unit tests will also be written to ensure the quality of the API.

The system will communicate with an existing SQL Server database, which is also used by many of Talley's other projects and services. This database may require minor modifications to the data, tables, and relationships to work with the new API and client systems.

1.4 IT Team & My Role

Talley's Information Technology department is primarily situated across the Motueka and Nelson offices, with the software developers based in Nelson. The core software development team consists of four developers: an intermediate developer, a senior developer, part-time senior developer, and a software architect. Note, the part-time senior was replaced by a second senior developer near the end of the project. There are also three testers who work alongside the developers, with one of them occasionally assisting with development tickets.

My role is intern/junior full-stack software developer, focussing solely on the migration of their legacy food processing data management system (RMS Admin). I originally began working on this migration project in a summer internship. I was drawn to this role as it was an ideal balance of my interests and strengths, along with technologies that are widely used and would be beneficial for me to gain more experience with.

The new system is being constructed using TypeScript React to build the client and C# .NET with Entity Framework for the API, to connect in with an existing SQL Server database. Through my studies, I enjoyed both website and software development, so this role would allow me to continue to develop my skills in both areas. I thoroughly enjoyed my introduction class into C#, so I was excited to experience a large-scale API system built with it, which from my previous knowledge is a common approach by many New Zealand businesses. I was also interested in this role because it would give me the opportunity to experience working in a medium-sized software team, as opposed to being the sole developer like in class-projects and most of my other work experience.

Talley's has wanted to upgrade RMS Admin as part of a current multi-phase IT overhaul, with the migration beginning several years ago, however with a small/medium sized team maintaining a collection of vital business systems, they have not had the workforce to sufficiently develop the new system. By hiring a new contracted worker who could focus solely on the migration, it would help them get the system up to a functional level where they could begin transitioning users and developers to focus on the new system.

2 Background

2.1 Legacy Systems

Legacy systems can be broadly defined as long-lived applications that are comprised of outdated software and/or hardware (Fanelli et al., 2016; Greenough & Worth, 2006; Wolfart et al., 2021). More specifically, these are software systems that significantly resist modification and expansion due to their obsolete technologies, but are business critical, posing a considerable risk to the business if they fail (Fanelli et al., 2016; Khadka et al., 2014). These systems can emerge primarily from obsolete technologies, but also from the loss of core developers and thus their understanding of the system, continued increase in code complexity, inherited codebases, externally contracted development, and frequent modifications to poorly structured code (Fanelli et al., 2016; Khadka et al., 2014). Legacy systems constitute of a considerable portion of global production systems, with a recent study suggesting more than eight-hundred billion lines of COBOL code alone are still in daily use (Bourne, 2022).

Legacy systems are often characterised by the reliance on outdated technologies and architectures (Duvvur, 2023). These systems often lack the flexibility, scalability, extensibility, and documentation that would allow the business to more easily adapt the system to new business requirements in an efficient and cost-effective manner (Bisbal et al., 1997). The burden of technical debt, emergence of security vulnerabilities, and accumulated operating inefficiencies poses an ever-increasing challenge to the innovation of legacy systems, and their long-term sustainability (Ogunwole et al., 2023). In today's highly competitive environment, it is vital for organisations to be able to constantly grow and adapt their business focus to remain successful (Bisbal et al., 1997).

Bisbal et al. (1997) discuss other challenges organisations face by legacy systems: usually these systems are running on obsolete hardware, which can be slow and expensive to maintain. Maintenance of the software systems themselves are generally expensive too, requiring significant resources due to lacking documentation, limitations of the technologies, and a lower understanding of the system's internal workings (Bisbal et al., 1997; Khadka et al., 2014; Wolfart et al., 2021). These faults lead to more time tracing faults and repairing issues (Bisbal et al., 1997). Expansion of the system with new functionality is often challenging due to the non-extensible and inflexible design (Bisbal et al., 1997). Integration with other systems is frequently more difficult due to an absence of clean interfaces, and the system may even be incompatible with modern and emerging technologies, such as cloud environments, artificial intelligence, and machine learning (Bisbal et al., 1997; Ogunwole et al., 2023). Older systems typically have more security vulnerabilities from the use of outdated technology, posing a higher threat of data breaches, ransomware attacks, or other cybersecurity attacks. Regulatory requirements like data protection laws also evolve over time, so failure to modernise legacy systems may lead to compliance violations, resulting in reputational damage and financial penalties (Ogunwole et al., 2023).

Bisbal et al. (1997) suggests that due to many businesses' reluctance in migrating to newer technologies, many businesses find themselves a 'catch 22' situation, where these systems are mission critical to the organisation's operations, but also significantly impede progress. This idea is repeated by other authors, highlighting the importance of these systems to their

businesses, while simultaneously restricting digital evolution, innovation, and agility (Fanelli et al., 2016; Ogunwole et al., 2023; Wolfart et al., 2021). This common theme highlights the importance of approaching the migration or modernisation process carefully to produce the best outcome for the organisation.

2.2 Master Data Management

Master data is the unification of core business objects used across different operational sectors of the business into a synchronised, consistent view of the core business entities (Loshin, 2010). Early enterprise systems were typically comprised of a collection of applications and data sets for each operational line in the organisation, which caused a duplication of varying core business objects like customers, products, employees, and locations across the business (Loshin, 2010). In the 1980s, with the popularisation of relational database management systems and rapid technological advancements to desktops, managing these separate objects accurately become much more challenging, leading to a push towards unifying these core business objects into “master objects”, and the conceptualisation of master data management (Loshin, 2010).

The primary benefit of master data is having consistent, accurate, and current core business data to meet different business needs across the whole organisation (Hikmawati et al., 2021). At a technical level, this results in reduced data errors and eradication of data redundancy for higher data quality (Pansara, 2021). At a business level, higher data quality can lead to improved stakeholder decision-making, improved customer service, cost savings, improved operational efficiency, increased productivity, comprehensive customer knowledge, consistent and accurate reporting, improved competitiveness, improved risk management, improved regulatory compliance, quicker results data queries, and simplified application development (Hikmawati et al., 2021; Loser et al., 2004; Loshin, 2010).

2.3 Migration & Modernisation

Legacy software modernisation can be defined as the “process of evolving existing software by replacing, redeveloping, reusing, or migrating the software components and platforms, when traditional maintenance practices can no longer achieve the desired system properties” (Khadka et al., 2014, p.1). The principal objectives of software modernisation are to reduce maintenance and operating costs, increase flexibility, and gain a clearer understanding of the system infrastructure (Fanelli et al., 2016; Khadka et al., 2014). In comparison, legacy software migration is the process of migrating a system to a new environment while retaining the functionality of the existing system but *without* it having to be completely redeveloped (Bisbal et al., 1997). These terms are often used interchangeably despite having different meanings.

In addition to reducing maintenance and operating costs, increasing flexibility, and improving system understanding, migration or modernisation projects can lead to other benefits (Fanelli et al., 2016; Khadka et al., 2014). These include reduced future development time and costs by using modern technologies with stronger support, improved maintainability through a better structured system, improved ease for adapting to business and user needs, minimised

complexity of the overall system through refactoring, greater scalability, better security, stricter compliance, more and easier integration of other services, and improved performance and reliability through newer technologies and redesigned logic (Mishra, 2020; Seacord et al., 2001; Wolfart et al., 2021). A migration or modernisation project requires an initial cost but is an investment for the future of the application and the long-term benefits gained.

There are many different target environments that may be selected to migrate a system to. Some systems may be migrated to cloud services, migrated into another architecture (e.g. from monolithic to micro-services), or modernised to the web, among many other alternatives (Sivagnana Ganesan & Chithralekha, 2017). The best environment for migration depends on what suits the system itself, the skills of the teams, and the rest of the IT environment at the organisation.

There are many different approaches to migrating an application, some of the most common being database-first, database-last, back box, and white box. Database-first, as the name suggests, involves migrating the database first then redeveloping the application to work with the new database, which can be performed incrementally but requires the migration of data to the new database first (Sivagnana Ganesan & Chithralekha, 2017). Black box modernisation involves examining the inputs and outputs of the system to understand its interfaces (Greplová, 2023; Seacord et al., 2001). While box works in contrast by examining the internal workings to understand the legacy system, which is much more time consuming (Greplová, 2023; Seacord et al., 2001). Another approach is the big bang methodology, which redevelops the legacy system from scratch in a modern architecture and platform to give full ability to modernise the system, however this is much more costly and time consuming than other approaches (Sivagnana Ganesan & Chithralekha, 2017).

Modernisation can come with challenges that the organisation should be aware of (Greplová, 2023). Modernisation projects can be a huge undertaking due to accumulated technical debt and system complexity (Greplová, 2023). Modernisation often requires substantial time and cost investments which some organisations may not be able to afford, and can result in significant business disruption due to changes in business process and retaining of users (Greplová, 2023). In addition, a modernised system may no longer be compatible with previous integrations (Greplová, 2023).

3 Placement report

The following chapter discusses the undertaking of the project, the approach taken, and the key outcomes. The technologies used in this project were already predetermined, however this approach is analysed compared to other popular technologies. Talley's development team operates under an agile-DevOps hybrid model with the Azure DevOps toolset, allowing the team to iteratively develop the company's internal systems. The core functionality and special features developed throughout the migration are outlined and explained, as well as the architectural patterns implemented. The use of VB6 in 2025 and observations of AI in the workplace are also briefly discussed.

3.1 Technologies

As the modernisation of RMS Admin had already begun prior to this project, the technologies of the new system had already been predetermined. While this limits approaches to developing the new system, this removed a significant portion of preparation that would have involved researching, analysing, and selecting appropriate technologies. Ultimately, this meant that the project could focus primarily on the development of the new system. As many of the decisions selecting technologies predate this project, the exact reasoning cannot be discussed, however benefits and drawbacks for the technologies chosen in relation to the system and its development are discussed.

3.1.1 React Client

React is a popular JavaScript library for building user interfaces for single-page applications (Geeks for Geeks, 2025). React uses a component-based architectural approach to allow developers to construct the application based on custom building blocks (Geeks for Geeks, 2025). As a library not a framework, React doesn't force a specific structure but gives freedom to the developers to design the application how they want (React, n.d.a). Many global companies including Netflix, Facebook, and Airbnb use React (Pujara, 2024). React uses a markup syntax called JavaScript XML (JSX), along with JavaScript or TypeScript to implement UI and logic (React, n.d.a).

Coming into this project, the React client had already been setup with a few pages implemented. While the client had little functionality, the complexity of establishing a robust foundation for the application had already been completed. This included establishing a project structure, implementing login functionality, implementing page navigation with a breadcrumb navigation bar, and setting up the API configuration. Having this work already completed allowed the project to focus solely on adding and improving item management sections.

React was a suitable library for this project because it promotes rapid enterprise-level development with its developer-friendly environment, reusability of in-application components, and many prebuilt components from the community (Angular Minds, 2025; Patadiya, 2023; Pujara, 2024). Its virtual document object model (virtual DOM) approach to rendering pages

allows for faster and more efficient rendering, improving performance of large applications (Angular Minds, 2025; Patadiya, 2023; Pujara, 2024). Having a simpler learning curve than some other competitors helps React stay accessible, especially to existing developers in the team with minimal React experience (Patadiya, 2023). React's component structure and code management can improve application scalability, while the unidirectional data flow can help streamline data in the application (Angular Minds, 2025; Patadiya, 2023; Pujara, 2024). Additionally, React has been around for a long time and has gathered an extensive community, providing more support to developers and indicating that the library will be around for a long time (Patadiya, 2023).

3.1.1.1 Material UI

Material UI (MUI) is a popular user interface (UI) component library designed to help developers create clean and intuitive user interfaces faster (Material UI, 2025). Utilising this library allowed pages to be constructed much easier than if every component had to be built and styled manually, especially with more complicated elements like data tables or toggle switches. MUI's components worked seamlessly with React's JSX to create clean and smooth user interfaces.

MUI components were implemented throughout the client, most notably to display tables of data, for text inputs, and for buttons. MUI's table component was used across all list pages and many editing forms to easily map objects into a clear, logical grid display (Material UI, n.d.b). MUI's button component was used across every page for navigation buttons, activating CRUD functionality (see Figure 1), or for other interactions, as it provides a polished button that can be easily customised to the required color theme or appearance (Material UI, n.d.a). The client also implements many of MUI's other components, including the text field, checkbox, select, radio group, and switch components in the editing forms.

Figure 1: *Fishing methods editing page toolbar with a selection of customised MUI buttons.*



There are many other popular component libraries often used in modern web clients, including Ionic, React Bootstrap, Ant Design, and Angular Material (Fryč, 2024; Salonen, 2023). As this project did not touch upon any other component libraries, no comparison can be made. However, MUI was an advantageous library to use, providing professional components to speed up development with the only drawback being a minor limitation of some components in niche use cases.

3.1.1.2 Toastify

React Toastify is an open-source package to easily add customisable and clean notifications to web applications (Khadra, 2025).

Some use cases of Toastify in the web client are to display to the user:

- An error when an API request fails (e.g. unauthorised request, see Figure 2: *Toastify error message from a failed API request. Figure 2*).
- An error when invalid values are submitted and client validation fails.
- An error when the parent editing page is loaded with an invalid ID.
- An information message when the editing form is creating a new item.
- A success message when an API request succeeds (e.g. item successfully updated, see Figure 3: *Toastify success message from a successful API request.*).

Figure 2: *Toastify error message from a failed API request.*

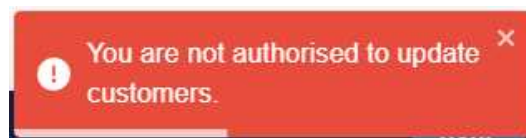
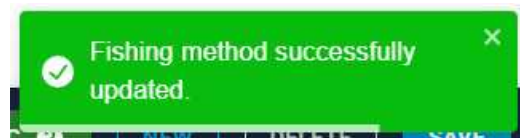


Figure 3: *Toastify success message from a successful API request.*



3.1.2 C# .NET API

Like with the React client, the initial C# .NET API framework had already been established, including a few controllers, services, schema, and models to facilitate the pages already constructed. Similarly, having a well-structured foundation made the process of expanding the API much simpler by eliminating any required setup. The API layer was however originally set up using Dapper as the object-relational mapper (ORM) and began transitioning to use Entity Framework Core shortly before this project began.

RESTful (representational state transfer) APIs have emerged as the standard for implementing modern APIs, especially for web applications like this project, which C# is capable of (Ehsan et al., 2022). C# is widely considered one of the most popular and capable languages for enterprise application and enterprise RESTful API development (Biri, 2024; Hakeem, 2024; Innocent, 2025; Marko, 2016; Pradhan, 2024). It is also one of the most popular languages in New Zealand, especially for enterprise-level applications, making it easier for Talley's to find relevantly knowledgeable and experienced programmers (TechSteps, n.d.).

C# is a widely used language, with a rich ecosystem of libraries, extensive community support,

and more experienced programmers, as well as supporting cross-platform development through .NET Core, compatibility with development accelerating tools like Swagger documentation generation, strong support for testing, good scalability, seamless integration with Visual Studio, and high performance (Biri, 2024; Hakeem, 2024; Innocent, 2025; Pradhan, 2024). Another benefit of using C#, as a Microsoft owned product, it has improved integration with Microsoft's SQL Server and many similarities to the legacy VB6 system, which will make the modernisation project simpler. Overall, C# is a strong, capable, and popular language suited to the development of enterprise applications/APIs, and was a good choice for this system.

3.1.2.1 *Entity Framework Core*

Entity Framework (EF) Core is an open-source, lightweight, extensible, cross-platform object-relational mapper (ORM) developed by Microsoft (Microsoft, 2024b). It is the reengineered replacement of Microsoft's Entity Framework and the recommended ORM for .NET (ZZZ Projects, 2024b). As an ORM, EF Core allows developers to work with a database using .NET objects, eliminating most of the data-access code that would typically be required (Microsoft, 2024b).

EF Core uses a context object that represents a session with the database, and a model comprised of entity classes to manage querying and saving changes to the database (Microsoft, 2024b). Instances of entity classes are retrieved from the database using C#'s built-in asynchronous Language Integrated Query (LINQ) features for simple use (Microsoft, 2024b). EF Core takes instances of models as input for creating, updating and deleting records from the database (Microsoft, 2023b).

Using an ORM is a beneficial design decision when working with an object-orientated programming (OOP) language like C#, because it allows developers to interact with the database using familiar object-oriented concepts, often leading to more efficient and less error-prone development, as well as easier maintenance of the codebase (ZZZ Projects, 2024b). Additionally, ORMs typically provide performance enhancing features such as caching, lazy loading, and connection pooling, which is important when developing modern large-scale applications (ZZZ Projects, 2024b). Furthermore, they provide a layer of abstraction between the application and database, making it easier to switch to a different database or improve scalability in the future (ZZZ Projects, 2024b).

Dapper is an open-source object-relational mapper (ORM) library for .NET applications (ZZZ Projects, 2024a). More specifically, Dapper is a micro-ORM: a cut-down, lightweight ORM designed for performance (ZZZ Projects, 2024a). The library allows for easy and quick access to data by executing raw SQL queries and mapping the results to objects to be used in the application, but offers none of the additional features often provided by other ORMs like EF Core (ZZZ Projects, 2024a).

An evaluation between EF Core and Dapper conducted by Forsberg (2022) concluded that for applications with the primary purpose of performance, Dapper is generally superior, however for general development, EF Core provides more additional features and better documentation for more efficient development. This claim is back up by Yaba (2023), who asserts that Dapper beats EF Core significantly in performance, which is its fundamental advantage, but may require a longer development period than other ORMs. Yaba (2023) reiterates that EF Core's

ability to automatically generate it's own SQL rather than requiring the developer to manually write it can simplify and speed up development, reducing overall cost. While both ORMs would be suitable for the development of Trace, the benefit of development ease and additional features offered by EF Core likely outweighs the raw performance increase from Dapper, especially as Trace is built for internal use by a smaller userbase and typically handles smaller quantities of data.

3.1.2.2 *Fluent API*

EF Core offers several primary approaches to configuring models: conventions, data annotations, and fluent API (Microsoft, 2023a). The fluent API configuration is considered the most powerful method and has the highest precedence, overriding the configurations of conventions and data annotations (Microsoft, 2023a). EF Core's fluent API configuration is based on the fluent interface design pattern, using method chaining to 'flow' between methods to create more readable, flexible, and concise code (Rout, n.d.).

3.1.2.3 *EF Core Power Tools*

EF Core Power Tools (Power Tools) is package that adds useful design-time and database features (ErikEJ, 2025). As this project already had a fully designed and constructed SQL Server database, Power Tools' reverse engineering feature allowed the automatic generation of models and schema directly from the existing tables (ErikEJ, 2025). This tool was very useful for saving development time when expanding the API, as it automated a significant portion of the setup process before writing the controllers and services.

3.1.2.4 *xUnit*

xUnit is an open source, unit testing tool for .NET (xUnit.net, n.d.). This project uses xUnit as a tool to ensure robustness and reliability of the API, and catch any breaking changes earlier in development through in-depth test coverage. As the API is expanded, both integration and unit tests are written or updated to maintain an extensive test coverage. Integration tests used to test the connection between two or more units, in this case that a controller correctly interacts with all of its invoked services, while unit tests focus on each unit of code, in this case each service method (PractiTest, 2024). xUnit supports two main types of tests: facts and theories. Facts are tests that are always true (see Figure 4), while theories take sets of data as an input and runs the test against each input (see Figure 5) (xUnit.net, n.d.). This makes it easier to test code with more test cases and boundary data or even assert for dynamic outcomes.

Figure 4: Example of an xUnit fact test from Trace.

```

1. [Fact]
2. public async Task DeleteBoatOwner_Exists_Success()
3. {
4.     // Arrange
5.     _unitTestContext.ContextBuilder.CreateBoatOwnerEntity();
6.
7.     // Act
8.     await _target.DeleteBoatOwner(1);
9.
10.    // Assert
11.    var owners = await _target.GetOwners();
12.
13.    // Assert
14.    Assert.DoesNotContain(1, owners.Select(x => x.OwnerId));
15. }
16.

```

Figure 5: Example of an xUnit theory test from Trace.

```

1. [Theory]
2. [InlineData(new[] { 1 }, "", 1, "No printer specified.")]
3. [InlineData(new int[] { 0 }, "testPrinter", 1, "No locations specified.")]
4. [InlineData(new[] { 1 }, "testPrinter", 0, "Cannot print less than one copy.")]
5. public async Task PrintLocationLabel_InvalidRequest_ThrowsException(
6.     int[] locationIds, string windowsPrinterName, int numberToPrint, string expectedMessage
7. )
8. {
9.     // Arrange
10.    SetupScenario();
11.
12.    // Act
13.    var ex = await Assert.ThrowsAsync<PrintFailedException>(
14.        () => Target.PrintLocationLabel(locationIds, windowsPrinterName, numberToPrint)
15.    );
16.
17.    // Assert
18.    Assert.Contains(expectedMessage, ex.Message);
19. }

```

3.1.3 SQL Server Database

SQL Server is a relational database management system (RDBMS) developed by Microsoft (Microsoft, 2024c). Trace operates over the existing SQL Server database used by the existing legacy system, which is filled with accumulated production data. While this eliminates the substantial work required to elicit requirements, design, and construct a new database, this limits the new system to having to work with this database. As the existing database has been in place for a long time, it has accumulated the bloat of unused tables, unused columns, duplicate columns, duplicate data, and invalid data, with the restriction of being used by live systems making it difficult to attempt to clean up and normalise the database. While some small adjustments were made throughout this project, including correcting some invalid data, updating columns to have matching key column types, and adding missing primary keys, a dedicated data cleansing project would be a beneficial future endeavour.

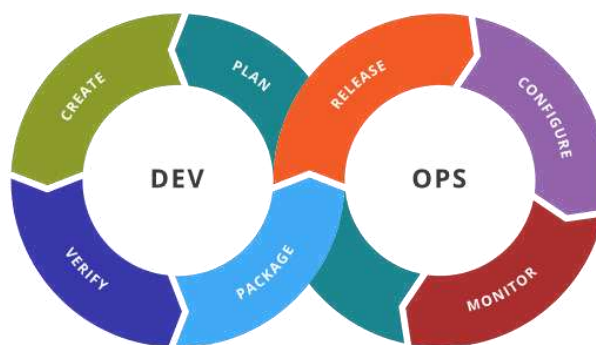
Figuroa et al. (2017) performed a comparison of SQL Server and PostgreSQL, another wide-

used enterprise-class RDBMS, concluding that while both include competitive features and support for common database activities, PostgreSQL's implementation was stricter to the SQL standard, whereas SQL Server favoured a more developer-friendly approach and included some features that would likely reduce development work (Figuerola et al., 2017). Another RDBMS is MySQL, which is a capable and cost-effective alternative but does not quite meet the same performance, advanced features, comprehensive security options, robustness, and enterprise-level support that SQL Server has (Spec India, 2024). Any of these popular RDBMSs would likely fulfil the needs of Talley's database system, with SQL Server likely being chosen because of the higher-level enterprise support and features offered which Talley's could have the option to make use of, along with it being a Microsoft product with enhanced compatibility with .NET.

3.2 Methodology, Pipeline & Processes

Talley's development team operates under an agile-DevOps hybrid model. Agile is an iterative methodology that breaks down projects into sections called sprints (Laoyan, 2025). After each sprint is complete, teams often reflect on if there was anything that could be improved upon and adjust their strategy for the next sprint to make sure development is as smooth and efficient as possible (Laoyan, 2025). This creates an agile development process, focussed on collaborative teamwork and meeting customer needs (Laoyan, 2025). DevOps has many similarities with Agile, but is instead a "set of practices, tools, and cultural philosophy that automate and integrate processes between software development and IT teams" (Atlassian, n.d.). This model emphasizes cross-team communication and collaboration to increase the speed and quality of software deployment (Atlassian, n.d.). The DevOps process incorporates tools to automate and accelerate processes, improving reliability and efficiency (Atlassian, n.d.). The iterative nature of DevOps is often represented by an infinity loop (see Figure 6) to demonstrate how each phase relates to each other and repeatedly cycles (Atlassian, n.d.). Talley's development process does not fully embrace DevOps principles but utilises the Azure DevOps toolset and implements a mix of both DevOps and agile practices. This hybrid methodology allows the development team to efficiently produce iterative improvements across their internal software and ship regular updates to facilitate constantly improving and expanding systems.

Figure 6: *DevOps life cycle.*



Note. From *DevOps toolchain* by Kharnagy, 2016, Wikimedia Commons (<https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg>). CC BY-SA 4.0.

3.2.1 DevOps Culture

DevOps has several core fundamental practices that promote team empowerment, cross-team collaboration, and technology automation (Atlassian, n.d.). Continuous integration is the practice of developers regularly merging new changes into the central code repository for automated building and testing (AWS, n.d.). This helps identify bugs quicker, improving software quality and reducing the time to validate new software releases (AWS, n.d.). Continuous delivery expands upon continuous integration (together called CI/CD) by automatically building, testing, and preparing software releases (AWS, n.d.). Properly implemented continuous delivery will always have a deployment-ready version available (AWS, n.d.). DevOps led applications often use a microservice architecture, as it allows a system to be built as a collection of smaller services, that can be operated and deployed independently, which helps facilitate CI/CD (Atlassian, n.d.). Another key concept is infrastructure as code, in which infrastructure is provisioned and managed using software development techniques including version control and continuous integration (AWS, n.d.). This code-based approach improves deployment efficiency, improves consistency, and allows better management of changes and versions (AWS, n.d.; IBM, 2021). Automation is one of the most important practices, and ties in with all of the other components of DevOps, automatically triggering builds, tests, and deployments to allow teams to develop high-quality software much faster (Atlassian, n.d.). Finally, monitoring all stages of the development lifecycle allows teams to respond to any issues quickly, and allows organisations to analyse how changes are affecting users (Atlassian, n.d.; AWS, n.d.).

These fundamentals promote several benefits to the development team and development output. Practicing DevOps can improve release frequency, quality, and stability, allowing deliverable goals to be met faster and giving the business a competitive advantage (Atlassian, n.d.). Increased collaboration between developers, testers, and operations members can make teams more efficient by reducing handoffs and environment consistency (Atlassian, n.d.). Practices like CI/CD help ensure that changes are functional and ready to be deployed, improving overall quality (Atlassian, n.d.).

3.2.2 Talley's Methodology

Talley's development operates under an agile-DevOps hybrid approach, pulling in principles and practices from both methodologies. The development team uses the Azure DevOps toolchain to manage the features, bugs, and tasks that need working on across their software projects. Azure Boards is used to track ticket progress across different development stages, as well as between development and testing teams (see Figure 7). Azure DevOps' built in wiki and repository sections help keep the management of all information and code centred in one place for easy management. Azure Pipelines is used to automate builds, testing, and deployment for improved efficiency. The senior developer of the team as a DevOps engineer to manage the pipeline and releases.

Talley's development team uses two types of meetings to keep the team connected. During daily standups, each developer briefly shares what they were working on yesterday and what they are working on now. This also gives team members an opportunity to raise issues that are blocking progress, allowing the team to easily discuss solutions or share important updates. Talley's also uses a weekly sprint planning meeting to review progress on the current sprint and discuss any new features or bugs that have been identified/submitted. These are both two common meetings used by agile development teams (Radigan, n.d.). The senior engineer acts as the team lead to facilitate both meetings, directing the flow of conversation and ensuring everything is covered.

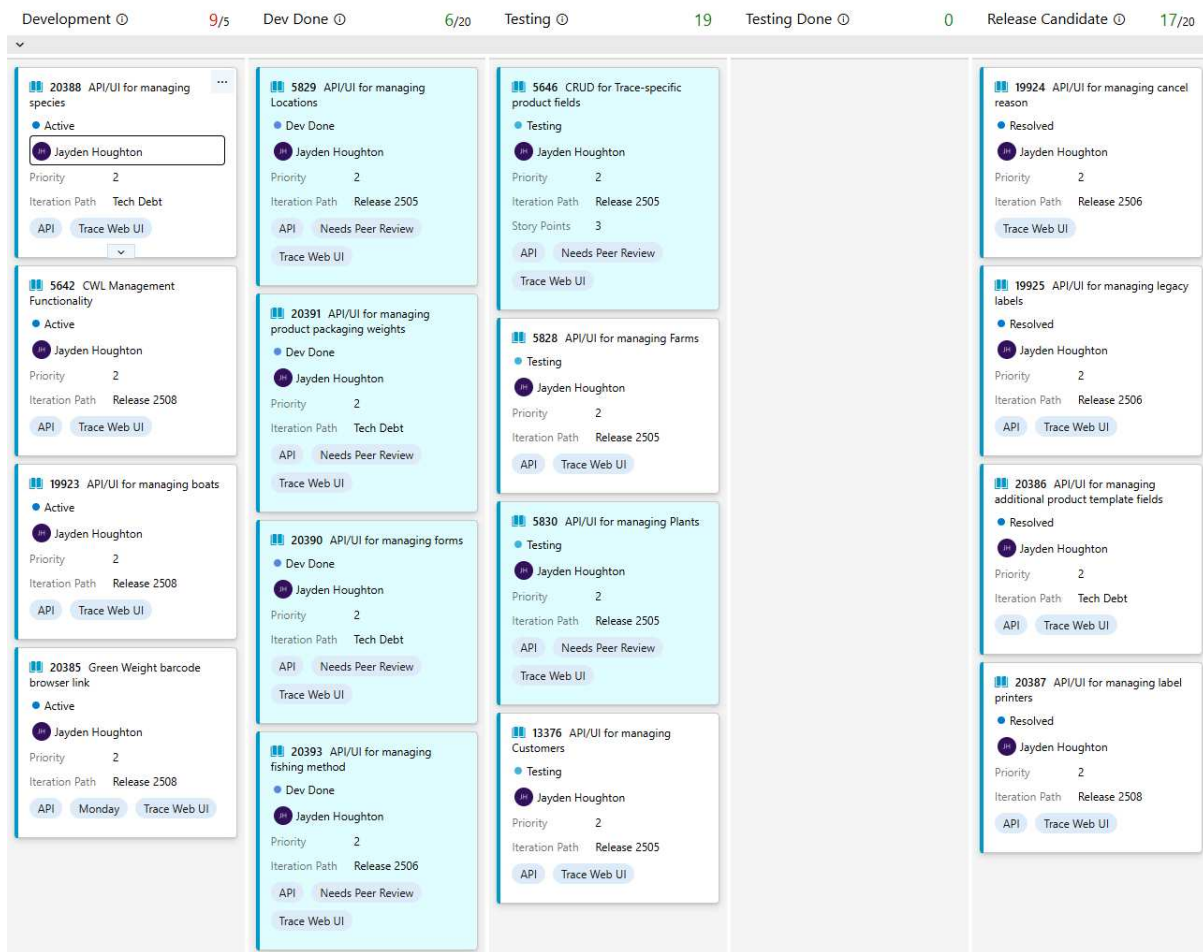
While Talley's does use Azure DevOps and implements some DevOps principles including cross-team communication, automation, daily standups, and sprint planning meetings, they don't implement CI/CD, opting for a slower release cycle of every two weeks and letting developers merge branches just when the task is complete. As CI/CD is an integral part of DevOps, this means Talley's development process is best described as an agile-DevOps approach. Having a hybrid approach allows the team to tailor the methodologies to best suit their development environment, fostering a more efficient and effective workflow (Das, 2024).

3.2.3 Development Pipeline & Tools

Azure DevOps is a collection of tools and services that form the centre point of project management and collaboration for Talley's development, testing and support teams. Azure Boards (see Figure 7) operates like a Kanban board, allowing team members to create tickets and progress them through columns, easily visualising team progress and the current development state (Microsoft, 2025e). Azure DevOps allows the creation of different work items to represent different types of work (Microsoft, 2024a). Epic and feature items represent larger work scenarios, while user stories and bugs are the main ticket types used to track work (Microsoft, 2024a). Azure Repos provides easy access, management, and version control of Git repositories (Microsoft, 2025e). Azure Pipelines is used to manage and automate builds, testing, and releases of applications (Microsoft, 2025e). Integration of all these tools into one platform provides a seamless management system for software development.

As a ticket is developed it moves through the following stages (see Figure 7):

- Backlog: A collection of tickets that have not been started.
- Development: All tickets currently being developed.
- Dev done: Tickets that a developer has finished. These are ready for peer review or may be undergoing peer review improvements.
- Testing: Tickets being tested independently by the testing team or undergoing fixes for discovered bugs.
- Testing done: Ticket is tested with no outstanding bugs.
- Release candidate (RC): Integration testing of all tickets for the upcoming release. Tickets may be going through development fixes for discovered issues.
- Release candidate done: Tickets that have passed RC testing and have no outstanding bugs.
- Live: Tickets that are in the live release.

Figure 7: DevOp's online ticket board showing in progress tickets for this project.

Peer reviewing is the process of having other developers check and review a piece of code (Kumari, 2024). At Talley's, all submitted pull requests are peer reviewed by at least one other developer (ideally more) before they are approved and moved to testing. By having other developers review the code, it helps ensure that the code meets its required criteria, bugs are found earlier in the development process, and possible improvements are highlighted (Kumari, 2024). These all lead to higher quality code and a smoother testing process.

Every ticket is then independently tested to ensure that it functions as intended and that this functionally fulfills the required criteria. To do this, the submitted branch is tested in-depth to find any issues or boundary cases that cause issues. Any discovered issues are passed on to the developer to fix them before the testing is repeated.

The release candidate is an internal build of all the new changes proposed for the next release (Teamhub, 2024). This build is thoroughly tested to identify any unresolved issues that have made it past the other tests or may be caused by the combination of new tickets, such as from conflicting changes (Teamhub, 2024). Any issues found are passed back to the related developer as priority work to fix, before the release candidate becomes the release build and is deployed to the live systems (Teamhub, 2024).

3.2.4 Research Process

Often throughout this project, there were many times when existing knowledge wasn't enough to be able to fully implement a feature or solve a bug. This meant that being able to research and find answer was integral to the success of the project. Research also gave the opportunity to practice problem solving and gain new knowledge of the technologies.

When encountered with an unknown bug or a gap in knowledge, the first step taken was to research via the internet. Depending on the topic in question, searches would often lead to official documentation or tutorial guides that would supply and explain the answers required. Stack Overflow (Stack Exchange Inc., n.d.) was one of the most useful websites, hosting a wide collection of questions and answers to all manner of programming issues, including more niche issues that had few other resources. A research attempt was always made first to reduce interruption of the other developer's work, and to practice problem solving.

If researching the solution was fruitless, the other developers in the office were the next resource for advice, guidance, and answers. The team's senior developer in particular had expansive knowledge and experience about the tools, technology, and system. Their guidance was especially useful for large, complex issues that were hard to explain or effectively research for. Often help from a developer would lead to another researching attempt but with a new direction to try, or a better understanding of the issue to search for.

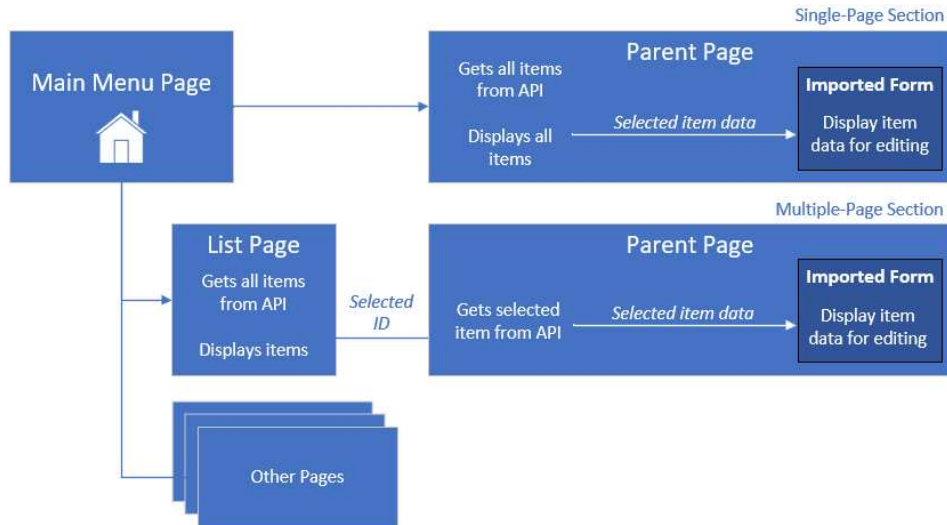
When other resources were insufficient, the artificial intelligence (AI) large language model (LLM) ChatGPT (OpenAI, 2024) was used to provide guidance to solutions. This tool was used sparingly and selectively as to not interfere with learning and demonstration of skills during this project. All uses are fully referenced in the appendix, including the prompt and model used (see Appendix A).

3.3 Client Structure

Trace is structured with a central landing page, with a navigation bar leading to each section. Each section of the program represents the management of a different 'entity', such as boats, species, or printer labels. Sections can be navigated to from any other section. Figure 8 demonstrates the structure visually.

Each section is typically comprised of two or three components:

- List page. This page displays the items for that section in a paginated format (divided into pages) (see Figure 9). Smaller sections often do not have this page, instead displaying a list of items in the modifying page.
- Editing/parent page. This acts as the parent page for the data editing form and holds the API calls. This page may also display a list of items in smaller sections (single-page sections) that have few items and few properties (see Figure 10).
- Editing form. This form is passed through the selected item's data from the parent page and handles the display of all properties. Submitting the form or similar actions are handled in the parent page (see Figures 9 and 10).

Figure 8: Structure of sections, pages, and forms in the client application.**Figure 9: Trace species list page.**

Talley's

Dev | Dev | JH.Admin

Home / Species

INVENTORY

ORDERS

MANAGE

Q

SIGN OUT

Species Name or Id

< 1 2 3 4 5 ... 43 >

Id	Description	Description AKA	Abbreviation	Whitefish
189	ABYSSAL HALOSAUR	ABYSSAL HALOSAUR	HAL	No
190	ABYSSAL RATTAIL	ABYSSAL RATTAIL	CMU	No
191	ACANTHEPHYRA PELAGICA	ACANTHEPHYRA PELAGICA	APE	No
192	ACANTHEPHYRA QUADRISPINOSA	ACANTHEPHYRA QUADRISPINOSA	AQU	No
193	ACANTHEPHYRA SPP.	ACANTHEPHYRA SPP.	ACA	No
194	ACHIROPSETTA TRICHOLEPIS	ACHIROPSETTA TRICHOLEPIS	ACT	No
19426	ADMIN			No
195	AETIDEID COPEPOD	AETIDEID COPEPOD	ZAE	No
196	AHURU	AHURU	PCO	No
1	ALBACORE TUNA	ALBACORE TUNA	ALB	No
197	ALBATROSS (UNIDENTIFIED)	ALBATROSS (UNIDENTIFIED)	XAL	No
198	ALDROVANDIA AFFINIS	ALDROVANDIA AFFINIS	ALA	No
199	ALERT PIGFISH	ALERT PIGFISH	API	No
2	ALFONSINO	ALFONSINO	BYX	No
200	ALL AND ANY UNIDENTIFIED SPECIES	ALL AND ANY UNIDENTIFIED SPECIES	UNX	No
201	AMBERJACK	AMBERJACK	AMB	No
202	AMBLYRAJA GEORGIANA	AMBLYRAJA GEORGIANA	SRR	No
203	AMPHIOXUS	AMPHIOXUS	ZAM	No
204	AMPHIPOD	AMPHIPOD	APH	No
205	AMPHIPOD - BENTHIC (GAMMARID)	AMPHIPOD - BENTHIC (GAMMARID)	APB	No
206	AMPHIPOD - PELAGIC (HYPERIID)	AMPHIPOD - PELAGIC (HYPERIID)	APP	No
207	ANCHOVY	ANCHOVY	ANC	No
208	ANEMONES	ANEMONES	ANT	No
209	ANGLED WEDGE SHELL	ANGLED WEDGE SHELL	PGA	No
210	ANONYMUS	ANONYMUS	ANX	No
211	ANONYMUS	ANONYMUS	ANX	No
212	ANONYMUS	ANONYMUS	ANX	No
213	ANONYMUS	ANONYMUS	ANX	No
214	ANONYMUS	ANONYMUS	ANX	No
215	ANONYMUS	ANONYMUS	ANX	No
216	ANONYMUS	ANONYMUS	ANX	No
217	ANONYMUS	ANONYMUS	ANX	No
218	ANONYMUS	ANONYMUS	ANX	No
219	ANONYMUS	ANONYMUS	ANX	No
220	ANONYMUS	ANONYMUS	ANX	No
221	ANONYMUS	ANONYMUS	ANX	No
222	ANONYMUS	ANONYMUS	ANX	No
223	ANONYMUS	ANONYMUS	ANX	No
224	ANONYMUS	ANONYMUS	ANX	No
225	ANONYMUS	ANONYMUS	ANX	No
226	ANONYMUS	ANONYMUS	ANX	No
227	ANONYMUS	ANONYMUS	ANX	No
228	ANONYMUS	ANONYMUS	ANX	No
229	ANONYMUS	ANONYMUS	ANX	No
230	ANONYMUS	ANONYMUS	ANX	No
231	ANONYMUS	ANONYMUS	ANX	No
232	ANONYMUS	ANONYMUS	ANX	No
233	ANONYMUS	ANONYMUS	ANX	No
234	ANONYMUS	ANONYMUS	ANX	No
235	ANONYMUS	ANONYMUS	ANX	No
236	ANONYMUS	ANONYMUS	ANX	No
237	ANONYMUS	ANONYMUS	ANX	No
238	ANONYMUS	ANONYMUS	ANX	No
239	ANONYMUS	ANONYMUS	ANX	No
240	ANONYMUS	ANONYMUS	ANX	No
241	ANONYMUS	ANONYMUS	ANX	No
242	ANONYMUS	ANONYMUS	ANX	No
243	ANONYMUS	ANONYMUS	ANX	No
244	ANONYMUS	ANONYMUS	ANX	No
245	ANONYMUS	ANONYMUS	ANX	No
246	ANONYMUS	ANONYMUS	ANX	No
247	ANONYMUS	ANONYMUS	ANX	No
248	ANONYMUS	ANONYMUS	ANX	No
249	ANONYMUS	ANONYMUS	ANX	No
250	ANONYMUS	ANONYMUS	ANX	No
251	ANONYMUS	ANONYMUS	ANX	No
252	ANONYMUS	ANONYMUS	ANX	No
253	ANONYMUS	ANONYMUS	ANX	No
254	ANONYMUS	ANONYMUS	ANX	No
255	ANONYMUS	ANONYMUS	ANX	No
256	ANONYMUS	ANONYMUS	ANX	No
257	ANONYMUS	ANONYMUS	ANX	No
258	ANONYMUS	ANONYMUS	ANX	No
259	ANONYMUS	ANONYMUS	ANX	No
260	ANONYMUS	ANONYMUS	ANX	No
261	ANONYMUS	ANONYMUS	ANX	No
262	ANONYMUS	ANONYMUS	ANX	No
263	ANONYMUS	ANONYMUS	ANX	No
264	ANONYMUS	ANONYMUS	ANX	No
265	ANONYMUS	ANONYMUS	ANX	No
266	ANONYMUS	ANONYMUS	ANX	No
267	ANONYMUS	ANONYMUS	ANX	No
268	ANONYMUS	ANONYMUS	ANX	No
269	ANONYMUS	ANONYMUS	ANX	No
270	ANONYMUS	ANONYMUS	ANX	No
271	ANONYMUS	ANONYMUS	ANX	No
272	ANONYMUS	ANONYMUS	ANX	No
273	ANONYMUS	ANONYMUS	ANX	No
274	ANONYMUS	ANONYMUS	ANX	No
275	ANONYMUS	ANONYMUS	ANX	No
276	ANONYMUS	ANONYMUS	ANX	No
277	ANONYMUS	ANONYMUS	ANX	No
278	ANONYMUS	ANONYMUS	ANX	No
279	ANONYMUS	ANONYMUS	ANX	No
280	ANONYMUS	ANONYMUS	ANX	No
281	ANONYMUS	ANONYMUS	ANX	No
282	ANONYMUS	ANONYMUS	ANX	No
283	ANONYMUS	ANONYMUS	ANX	No
284	ANONYMUS	ANONYMUS	ANX	No
285	ANONYMUS	ANONYMUS	ANX	No
286	ANONYMUS	ANONYMUS	ANX	No
287	ANONYMUS	ANONYMUS	ANX	No
288	ANONYMUS	ANONYMUS	ANX	No
289	ANONYMUS	ANONYMUS	ANX	No
290	ANONYMUS	ANONYMUS	ANX	No
291	ANONYMUS	ANONYMUS	ANX	No
292	ANONYMUS	ANONYMUS	ANX	No
293	ANONYMUS	ANONYMUS	ANX	No
294	ANONYMUS	ANONYMUS	ANX	No
295	ANONYMUS	ANONYMUS	ANX	No
296	ANONYMUS	ANONYMUS	ANX	No
297	ANONYMUS	ANONYMUS	ANX	No
298	ANONYMUS	ANONYMUS	ANX	No
299	ANONYMUS	ANONYMUS	ANX	No
300	ANONYMUS	ANONYMUS	ANX	No
301	ANONYMUS	ANONYMUS	ANX	No
302	ANONYMUS	ANONYMUS	ANX	No
303	ANONYMUS	ANONYMUS	ANX	No
304	ANONYMUS	ANONYMUS	ANX	No
305	ANONYMUS	ANONYMUS	ANX	No
306	ANONYMUS	ANONYMUS	ANX	No
307	ANONYMUS	ANONYMUS	ANX	No
308	ANONYMUS	ANONYMUS	ANX	No
309	ANONYMUS	ANONYMUS	ANX	No
310	ANONYMUS	ANONYMUS	ANX	No
311	ANONYMUS	ANONYMUS	ANX	No
312	ANONYMUS	ANONYMUS	ANX	No
313	ANONYMUS	ANONYMUS	ANX	No
314	ANONYMUS	ANONYMUS	ANX	No
315	ANONYMUS	ANONYMUS	ANX	No
316	ANONYMUS	ANONYMUS	ANX	No
317	ANONYMUS	ANONYMUS	ANX	No
318	ANONYMUS	ANONYMUS	ANX	No
319	ANONYMUS	ANONYMUS	ANX	No
320	ANONYMUS	ANONYMUS	ANX	No
321	ANONYMUS	ANONYMUS	ANX	No
322	ANONYMUS	ANONYMUS	ANX	No
323	ANONYMUS	ANONYMUS	ANX	No
324	ANONYMUS	ANONYMUS	ANX	No
325	ANONYMUS	ANONYMUS	ANX	No
326	ANONYMUS	ANONYMUS	ANX	No
327	ANONYMUS	ANONYMUS	ANX	No
328	ANONYMUS	ANONYMUS	ANX	No
329	ANONYMUS	ANONYMUS	ANX	No
330	ANONYMUS	ANONYMUS	ANX	No
331	ANONYMUS	ANONYMUS	ANX	No
332	ANONYMUS	ANONYMUS	ANX	No
333	ANONYMUS	ANONYMUS	ANX	No
334	ANONYMUS	ANONYMUS	ANX	No
335	ANONYMUS	ANONYMUS	ANX	No
336	ANONYMUS	ANONYMUS	ANX	No
337	ANONYMUS	ANONYMUS	ANX	No
338	ANONYMUS	ANONYMUS	ANX	No
339	ANONYMUS	ANONYMUS	ANX	No
340	ANONYMUS	ANONYMUS	ANX	No
341	ANONYMUS	ANONYMUS	ANX	No
342	ANONYMUS	ANONYMUS	ANX	No
343	ANONYMUS	ANONYMUS	ANX	No
344	ANONYMUS	ANONYMUS	ANX	No
345	ANONYMUS	ANONYMUS	ANX	No
346	ANONYMUS	ANONYMUS	ANX	No
347	ANONYMUS	ANONYMUS	ANX	No
348	ANONYMUS	ANONYMUS	ANX	No
349	ANONYMUS	ANONYMUS	ANX	No
350	ANONYMUS	ANONYMUS	ANX	No
351	ANONYMUS	ANONYMUS	ANX	No
352	ANONYMUS	ANONYMUS	ANX	No
353	ANONYMUS	ANONYMUS	ANX	No
354	ANONYMUS	ANONYMUS	ANX	No
355	ANONYMUS	ANONYMUS	ANX	No
356	ANONYMUS	ANONYMUS	ANX	No
357	ANONYMUS	ANONYMUS	ANX	No
358	ANONYMUS	ANONYMUS	ANX	No
359	ANONYMUS	ANONYMUS	ANX	No
360	ANONYMUS	ANONYMUS	ANX	No
361	ANONYMUS	ANONYMUS	ANX	No
362	ANONYMUS	ANONYMUS	ANX	No
363	ANONYMUS	ANONYMUS	ANX	No
364	ANONYMUS	ANONYMUS	ANX	No
365	ANONYMUS	ANONYMUS	ANX	No
366	ANONYMUS	ANONYMUS	ANX	No
367	ANONYMUS	ANONYMUS	ANX	No
368	ANONYMUS	ANONYMUS	ANX	No
369	ANONYMUS	ANONYMUS	ANX	No
370	ANONYMUS	ANONYMUS	ANX	No
371	ANONYMUS	ANONYMUS	ANX	No
372	ANONYMUS	ANONYMUS	ANX	No
373	ANONYMUS	ANONYMUS	ANX	No
374	ANONYMUS	ANONYMUS	ANX	No
375	ANONYMUS	ANONYMUS	ANX	No
376	ANONYMUS	ANONYMUS	ANX	No
377	ANONYMUS	ANONYMUS	ANX	No
378	ANONYMUS	ANONYMUS	ANX	No
379	ANONYMUS	ANONYMUS	ANX	No
380	ANONYMUS	ANONYMUS	ANX	No
381	ANONYMUS	ANONYMUS	ANX	No
382	ANONYMUS	ANONYMUS	ANX	No
383	ANONYMUS	ANONYMUS	ANX	No
384	ANONYMUS	ANONYMUS	ANX	No
385	ANONYMUS	ANONYMUS	ANX	No
386	ANONYMUS	ANONYMUS	ANX	No
387	ANONYMUS	ANONYMUS	ANX	No
388	ANONYMUS	ANONYMUS	ANX	No
389	ANONYMUS	ANONYMUS	ANX	No
390	ANONYMUS	ANONYMUS	ANX	No
391	ANONYMUS	ANONYMUS	ANX	No
392	ANONYMUS	ANONYMUS	ANX	No
393	ANONYMUS	ANONYMUS	ANX	No
394	ANONYMUS	ANONYMUS	ANX	No
395	ANONYMUS	ANONYMUS	ANX	No
396	ANONYMUS	ANONYMUS	ANX	No
397	ANONYMUS	ANONYMUS	ANX	No
398	ANONYMUS	ANONYMUS	ANX	No
399	ANONYMUS	ANONYMUS	ANX	No
400	ANONYMUS	ANONYMUS	ANX	No
401	ANONYMUS	ANONYMUS	ANX	No
402	ANONYMUS	ANONYMUS	ANX	No
403	ANONYMUS	ANONYMUS	ANX	No
404	ANONYMUS	ANONYMUS	ANX	No
405	ANONYMUS	ANONYMUS	ANX	No
406	ANONYMUS	ANONYMUS	ANX	No
407	ANONYMUS	ANONYMUS	ANX	No
408	ANONYMUS	ANONYMUS	ANX	No
409	ANONYMUS	ANONYMUS	ANX	No
410	ANONYMUS	ANONYMUS	ANX	No
411	ANONYMUS	ANONYMUS	ANX	No
412	ANONYMUS	ANONYMUS	ANX	No
413	ANONYMUS	ANONYMUS	ANX	No
414	ANONYMUS	ANONYMUS	ANX	No
415	ANONYMUS	ANONYMUS	ANX	No
416	ANONYMUS	ANONYMUS	ANX	No
417	ANONYMUS	ANONYMUS	ANX	No
418	ANONYMUS	ANONYMUS	ANX	No
419	ANONYMUS	ANONYMUS	ANX	No
420	ANONYMUS	ANONYMUS	ANX	No
421	ANONYMUS	ANONYMUS	ANX	No
422	ANONYMUS	ANONYMUS	ANX	No
423	ANONYMUS	ANONYMUS	ANX	No
424	ANONYMUS	ANONYMUS	ANX	No
425	ANONYMUS	ANONYMUS	ANX	No
426	ANONYMUS	ANONYMUS	ANX	No
427	ANONYMUS	ANONYMUS	ANX	No
428	ANONYMUS	ANONYMUS	ANX	No
429	ANONYMUS	ANONYMUS	ANX	No
430	ANONYMUS	ANONYMUS	ANX	No
431	ANONYMUS	ANONYMUS	ANX	No
432	ANONYMUS	ANONYMUS	ANX	No
433	ANONYMUS	ANONYMUS	ANX	No
434	ANONYMUS	ANONYMUS	ANX	No
435	ANONYMUS	ANONYMUS	ANX	No
436	ANONYMUS	ANONYMUS	ANX	No
437	ANONYMUS	ANONYMUS	ANX	No
438	ANONYMUS	ANONYMUS	ANX	No
439	ANONYMUS	ANONYMUS	ANX	No
440	ANONYMUS	ANONYMUS	ANX	No
441	ANONYMUS	ANONYMUS	ANX	No
442	ANONYMUS	ANONYMUS	ANX	No
443	ANONYMUS	ANONYMUS	ANX	No
444	ANONYMUS	ANONYMUS	ANX	No
445	ANONYMUS	ANONYMUS	ANX	No
446	ANONYMUS	ANONYMUS	ANX	No
447	ANONYMUS	ANONYMUS	ANX	No
448	ANONYMUS	ANONYMUS	ANX	No
449	ANONYMUS	ANONYMUS	ANX	No
450	ANONYMUS	ANONYMUS	ANX	No
451	ANONYMUS	ANONYMUS	ANX	No
452	ANONYMUS	ANONYMUS	ANX	No

NEW

Figure 10: Trace species editing page with editing form.

The screenshot shows the 'Species Details' editing form for 'ABYSSAL HALOSAUR'. The form includes fields for Description, Abbreviation, Species AKA (Yield), Quota (CLR) Abbreviation, Quota Type, Discard (CLR) Abbreviation, and GW Catch Area. A 'Use for Whitefish' checkbox is also present. Below the details is a 'Licenses' section with a table of available licenses.

ID	Description
<input type="checkbox"/> 0	LICENSE_FISH
<input type="checkbox"/> 1	LICENSE_MUSSEL
<input type="checkbox"/> 2	LICENSE_FPROD
<input type="checkbox"/> 3	LICENSE_ICECREAM
<input type="checkbox"/> 4	LICENSE_VESSEL
<input type="checkbox"/> 5	LICENSE_MUSSELSCALES
<input type="checkbox"/> 6	LICENSE_VEGE
<input type="checkbox"/> 7	LICENSE_DRYGOODS

At the bottom right, there are 'DELETE' and 'SAVE' buttons.

Note: The species section is a two-page section. The list of all species is displayed on a separate page.

Figure 11: Trace product fishing methods editing page with editing form.

The screenshot shows the 'Product Fishing Methods' editing page. It features a table of fishing methods and a 'Product Fishing Method Details' form for editing the selected method.

Fishing Method	Code	Active
Bottom longlining	BLL	Active
Bottom trawl	BT	Active
Bottom trawl pair	BPT	Inactive
Cod potting	CP	Inactive
Dredging	D	Inactive
Fish traps	FP	Inactive
Inshore drift netting	DN	Inactive
Mechanical harvesting	MH	Inactive
Midwater trawl	MW	Active
Midwater trawl pair	MPT	Inactive
Pole and line	PL	Inactive
Potting	POT	Active
Purse seining	PS	Inactive
Scoop nets	SCN	Inactive
Set netting	SN	Active
Surface longlining	SLL	Inactive
Trolling	T	Inactive

The 'Product Fishing Method Details' form for 'Bottom longlining' includes fields for Method Name, Code, and a checked 'Is Active' checkbox.

At the bottom, there are 'SYNC', 'NEW', 'DELETE', and 'SAVE' buttons.

Note: The product fishing methods section is a single-page section, thus it includes the fishing methods list and editing form in the same screen.

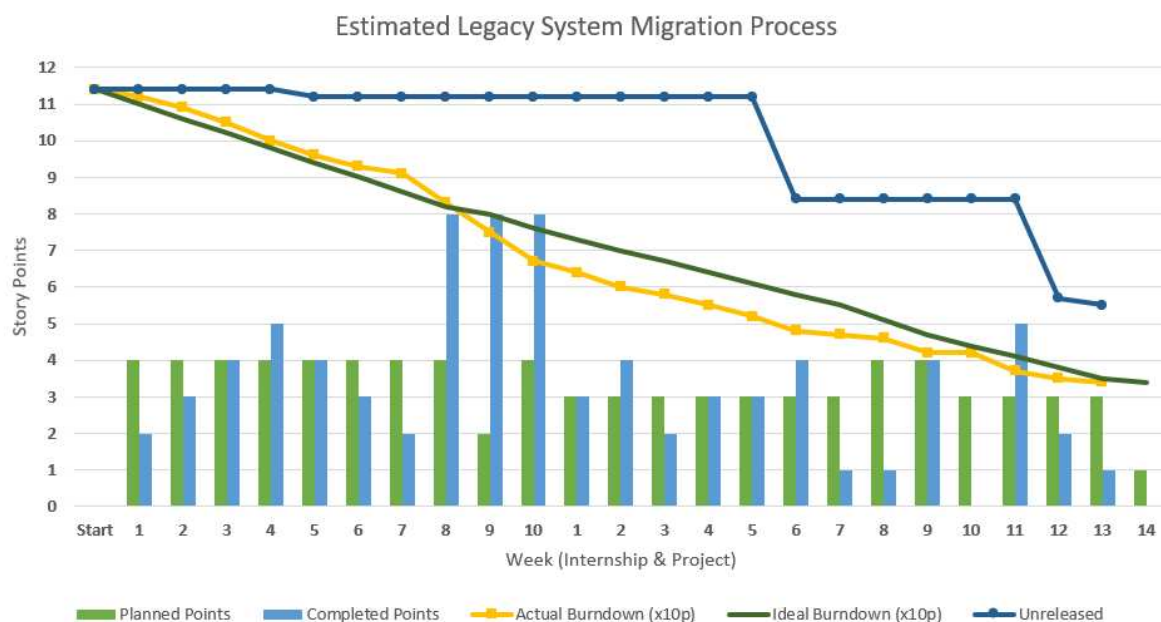
The benefit of these two types is that a list page allows for lots of items to be displayed easily, along with several properties in a table format, which suits entities like boats. With smaller entities like stacking patterns, where there are only a dozen items with a single property each, displaying them and editing them from the same page is much easier.

3.4 Completed Work

Over the course of the modernisation project, 26 sections were implemented, with an estimated value of 80 story points (see Table 1). Despite slightly less hours being completed than planned (see section 4.2 and Table 3), this perfectly aligns with the estimated completion of 80 story points illustrated by the planned points series in the project burndown chart (see Figure 12), which represents the ideal burndown. The original project proposal Made to Talley's was to have 50-70% of the system migrated by the end of the project, with the ideal burndown showing 70%. The project was finished with exactly 70% of story points completed, and 70% of tickets migrated. As of completing the project, 52% of the total story points have been deployed to the live release (see Figure 12).

With the project conclusion having successfully migrated the upper threshold of estimated progress, this project can be considered successful. The ideal outcome was achieved in slightly less work hours than originally planned and exactly on time, closely aligning to the initial targets, demonstrating effective planning and execution. This success is evident by the quantifiable results presented by Figure 12 and Table 1.

Figure 12: Burndown chart demonstrating the work completed over the project's duration.



Note. Tickets are represented by story points, which reflects the estimated work size for a more accurate representation of progress. The series lines shown in the chart are scaled to 10 story points (8 = 80 story points) to match appropriate scaling with the vertical columns. This chart

includes both the summer internship and the capstone project sections of the project. This chart also only includes the weeks where placement work was completed; the week numbers do not align directly to the timelines shown in Table 3).

Table 1 presents the list of sections migrated over the course of the project, including the designated story point value. Story points are a common technique in agile project management to estimate the effort required to complete a task (Vige, 2025). As the feature list for the migration was developed and expanded, estimated story point values were assigned to each section. For this project, one story point roughly equates to one day's work of developing the ticket, working through feedback, and writing tests.

Table 1: *List of sections migrated during the project and their estimated story point value.*

Number	Section	Story Points
1	Products (finalising)	4
2	Farms	4
3	Plants	4
4	Locations	5
5	Cancel reasons	3
6	Legacy labels	3
7	Additional product fields	3
8	Label printers	3
9	Product packaging weights	3
10	Product forms	3
11	Fishing methods	3
12	Customers	4
13	Species	4
14	Boats	4
15	CWL management (system & station admin pages)	2
16	Product group storage	3
17	Stack patterns	2
18	Metal detector presets	2
19	Boat captains	3
20	Ports of discharge	2
21	Vessel captains	3
22	Roles	2
23	Accounts	3
24	Boat owner	2
25	Product outers	2
26	Transporters	2
M	Miscellaneous bugfixes	2
Total	Story Points	80

Note. The list of sections is arranged in approximate order of completion. Each story point value is an estimated representation of the work that ticket will require.

3.4.1 CRUD Functionality

While many sections were implemented to allow the management of various entities, at its core, each section just implements a selection of basic create, read, update, and delete (CRUD) functionality. These functionalities work differently between one-page and two-page structured sections. At a fundamental level, buttons in the UI trigger API calls from the editing container page, which causes the API to process the request and update the database. Error handling with try catch blocks and error checking is used throughout the client and API to ensure that the requests are handled accurately.

3.4.1.1 Read

There are four points in the section at which the client may request data from the API:

1. When loading the list page to retrieve a paginated list of all or some items.
2. When loading the parent page to retrieve a list of all items.
3. When loading the parent page to retrieve the item for editing.
4. When loading the editing form as apart of retrieving additional editing data.

All except the editing data request work the same way, by triggering a method in the list page or parent page, and sending a request to the API. The request is routed to the correct controller and service, which uses LINQ to query the desired items from EF Core, returning the list to the client for display. The process for getting paginated results for the list page is slightly more complex, with the controller creating a search request object to help the service produce the correct results selection.

The two-page retrieval is comprised of four functions and `useEffects` (see Figure 13). The API request is wrapped in a `useCallback()` React Hook, which caches the function between re-renders, optimising performance (React, n.d.b). Since users can input an alphanumeric phrase to filter the API's search, a debouncing has been implemented with a `useDebounce()` method and a callback function. This stops the API from sending a request after every character the user types, instead waiting until their whole search phrase is entered, reducing API calls and optimising the application's performance (Ihnatovich, 2024).

Figure 13: Example of data retrieval in the React client from a two-page section.

```
1. const setFilterOptions = (callback: () => void): void => {
2.   callback();
3.   performFilterDebounce();
4. };
5.
6. const performFilter = useCallback(async () => {
7.   const boats = await api.boat.boatSearchGet(
8.     name,
9.     inactive,
10.    currentPage,
11.    pageSize,
12.  );
13.
14.   setItems(boats.data);
15.   setPages(boats.pageCount);
16.   setActive(inactive);
17. }, [name, inactive, currentPage, pageSize, api.boat]);
18.
19. const performFilterDebounce = useDebounce(() => {
20.   performFilter();
21. });
22.
23. useEffect(() => {
24.   performFilter();
25. }, [currentPage, performFilter]);
26.
```

For single-page section, there is no search meaning the API request can be called directly, however the `useEffect` must handle maintaining which item is selected (see Figure 14). This is done by storing a list of all items in a React `useState`, as well as storing the currently selected item in another `useState`. When data is being loaded, if the selected item is not set, the first item is selected by default, otherwise the IDs are matched to update the selected item. This ensures that the same item is selected even after creating or updating it.

Figure 14: Example of data retrieval in the React client from a single-page section.

```
1. useEffect(() => {
2.   const fetchFishingMethods = async () => {
3.     try {
4.       const res = await api.boat.boatGetFishingMethodsGet();
5.       setFishingMethods(res);
6.       if (res) {
7.         if (fishingMethod === null) {
8.           setFishingMethod(res![0]);
9.         } else {
10.          setFishingMethod(res.find(x => x.id === fishingMethod.id) || null);
11.        }
12.      }
13.      setIsCreate(false);
14.    } catch (error: unknown) {
15.      const errorJson = await ApiError.create(error as Response);
16.      toast.error(errorJson.message);
17.    }
18.  }
19.  if (submitSuccess) {
20.    fetchFishingMethods();
21.    setSubmitSuccess(false);
22.  }
23. }, [submitSuccess, api.boat, isCreate]);
24.
```

As well as the object being edited, some editing forms require additional data, such as option lists to populate dropdowns. Each section that needs this data has a separate editing data and editing data context files. The editing data context file defines a context tag used to wrap the editing form in the parent page's JSX. This allows the editing form to access the data variables defined in this context. The editing data file sends a request to the related 'get editing data' endpoint in the API, which retrieves all the required data from various services (see Figure 15 line 2). The API response is then processed into each of the variables, including any mapping to appropriate data types (e.g. list of items is converted to an ID-name pair for a dropdown) (see Figure 15).

Figure 15: Example of requesting and processing editing data.

```
1. const makeRequest = async () => {
2.   const response = await api.boat.boatGetDataForBoatEditingGet();
3.   setSkipperList([emptyOption, ...
4.     response.skippers.map((x) => {
5.       return {
6.         id: x.skipperId,
7.         label: x.skipperName,
8.       } as DropdownOption;
9.     }]),
10.  ]);
11.  // Other data
```

The item data context file creates the context for the editing data, including the *useItemData()* method to request the data in a form. In addition, the context specified what data is available with a TypeScript interface (see Figure 16). This file is tightly coupled with the item data file.

Figure 16: Example of an item data context interface.

```
1. interface BoatData {
2.   skipperList: DropdownOption[];
3.   ownerList: DropdownOption[];
4.   fishingMethodList: DropdownOption[];
5. }
```

If the data context tag is included in the parent page's JSX surrounding the editing form tag, the editing form can access this data. This can be done with the *useItemData()* method exported from the item data context file (see Figure 17). Any number of the exported variables can be retrieved depending on the requirements of the page.

Figure 17: Example of getting editing data in an editing form.

```
1. const {
2.   skipperList,
3.   ownerList,
4.   fishingMethodList
5. } = useBoatData();
```

3.4.1.2 Update

The save (update) button is in the editing form. Some business entities are only permitted to be managed from the Motueka site, as this is where their head office is situated and is their ‘master management location’. For these sections, the save button is disabled if the user is signed into any other locations.

All forms have basic validation to ensure that required fields have been entered. If this fails, a Toastify pop-up error is shown. After successful validation, the form submits the updated object to the parent page, which handles most of the client’s logic and sometimes performs additional validation, like checking if the new name is already in use (see Figure 18 lines 2-7). The parent page, then sends an update request with the object to the API. In the API, a ‘create or update’ method retrieves the existing data object and replaces the data with the new object’s data, saving it back to the database context.

Some objects include sub-objects (e.g. products have dynamic additional fields stored in another table), which are managed in the same page. When updating the main object, these secondary objects must be updated too. This is done by retrieving a list of existing items from the database to compare against the new list; any not in the new list are deleted, any not in the old list are added, and the rest are updated.

Figure 18: Example of a create or update API call including client validation.

```
1. const onSubmit = async (data: FishingMethod) => {
2.   if (isCreate) {
3.     if (fishingMethods?.some(x => x.fishingMethodName === data.fishingMethodName)) {
4.       toast.error("Fishing method already exists.");
5.       return;
6.     }
7.   }
8.
9.   try {
10.    const updatedMethod = await api.boat.boatCreateOrUpdateFishingMethodPost(data);
11.    toast.success("Fishing method successfully updated.");
12.    setSubmitSuccess(true);
13.    setFishingMethod(updatedMethod);
14.  } catch (error: unknown) {
15.    const errorJson = await ApiError.create(error as Response);
16.    toast.error(errorJson.message);
17.  }
18. };
```

3.4.1.3 Create

Create functionality is handled in two different ways in the client depending on the section’s structure. For both types, if the business entities are only permitted to be managed from the Motueka site, the create button will be disabled if the user is signed into any other locations. In a two-page section, creation is called from the list page. Instead of sending a valid item ID to the parent page and loading the editing form with an existing item, a “new” keyword is passed through as the route parameter. On receiving this value, the parent page will instead pass an empty object into the editing form, which can be filled out with the details of the new object

(see Figure 19 lines 5-11). The editing form is also passed a “isCreating” value, which dynamically changes some elements, including toggling some name and ID inputs as read-only when updating but editable when creating, or changing the save button to say “create” for a more intuitive user experience. In single-page sections there is no list page, so the create button simply sets the parent form’s selected object directly to an empty object, which is passed through to the form.

Figure 19: Example of two-page section editing (parent) page in requesting item data.

```
1. useEffect(() => {
2.   const fetchBoats = async () => {
3.     setNoResult(false);
4.     const boats = await api.boat.boatSearchGet(boatName, undefined, undefined, undefined);
5.     if (boatName === "new") {
6.       toast.info(`Creating new boat.`);
7.       setBoat({
8.         // Set undefined properties
9.       } as Boat);
10.      setIsCreate(true);
11.    }
12.    else if (!boats.data || boats.data.length === 0) {
13.      setNoResult(true);
14.      toast.error(`Boat ${boatName} not found.`);
15.      throw new ClientError(`Boat ${boatName} not found.`);
16.    } else {
17.      setBoat(boats.data[0]);
18.      setIsCreate(false);
19.    }
20.  };
21.  fetchBoats();
22. }, [state, boatName, api.boat]);
23.
```

When the create button is clicked, an in-form handler first performs client validation to ensure that required inputs are entered and valid. If this fails, a Toastify pop-up error is shown. On a success, the item’s updated data is passed back to the parent page, which handles most of the logic. Some pages also include additional validation, such as checking that the new unique name is not already in use (see Figure 18 lines 2-7). The API’s save request is then called.

Almost all sections have a combined ‘create or update’ method which will perform either depending on if the identifier (ID/name) exists in the database (see Figure 20). There are a couple of exceptions however, where pages have separate create and update methods. This was done to facilitate pages where there was no way to distinguish if the supplied object was new or updated. For example, the ID of plants is manually set by the user. If a user tried to create a new plant with an existing ID, the API cannot identify if the object is new, or if the user just changed the details, so it will always update the object instead of returning an error stating that the ID is already in use. Either way, the method will validate that the object includes all required properties. The object is added to the database context via EF Core, as well as any sub-objects it may include. The created object is then returned to the client, so the newly created item can be automatically selected from the list.

Figure 20: Example of a *CreateOrUpdateItem* API method using EF Core.

```
1. public async Task<FishingMethod> CreateOrUpdateFishingMethod(FishingMethod newFishingMethod)
2. {
3.     await using var dbContext =
4.         await _contextFactory.CreateDbContextAsyncWithDefaultConnection(_dbConnections);
5.
6.     var updatedFishingMethodEntity = _mapper.Map<FishingMethodEntity>(newFishingMethod);
7.     var existingFishingMethodEntity = await dbContext.FishingMethod.SingleOrDefaultAsync(
8.         p => p.Id == updatedFishingMethodEntity.Id);
9.
10.    try
11.    {
12.        if (existingFishingMethodEntity != null)
13.        {
14.            dbContext.Entry(existingFishingMethodEntity)
15.                .CurrentValue.SetValues(updatedFishingMethodEntity);
16.        }
17.        else
18.        {
19.            if (string.IsNullOrEmpty(updatedFishingMethodEntity.FishingMethodName))
20.            {
21.                throw new InvalidRequestException("FishingMethodName is a required field.");
22.            }
23.
24.            updatedFishingMethodEntity.Id = Guid.NewGuid();
25.            await dbContext.FishingMethod.AddAsync(updatedFishingMethodEntity);
26.        }
27.
28.        await dbContext.SaveChangesAsync();
29.        return _mapper.Map<FishingMethod>(updatedFishingMethodEntity);
30.    }
31.    catch (Exception ex)
32.    {
33.        _logger.LogError(ex,
34.            $"Failed to create or update fishing method {updatedFishingMethodEntity.Id}");
35.        throw;
36.    }
37. }
```

3.4.1.4 Delete

For both single-page and two-page sections, the delete button is in the editing form, and sends the current objects ID to the parent page. If a new item is being created, the button is disabled as there is no item to delete. The parent page sends the ID in a delete HTTP request to the API, which deletes the object from the database (see Figure 21).

Figure 21: Example of an API delete method using EF Core.

```
1. public async Task DeleteFishingMethod(Guid guid)
2. {
3.     await using var context =
4.         await _contextFactory.CreateDbContextAsyncWithDefaultConnection(_dbConnections);
5.
6.     var methodToDelete =
7.         await context.FishingMethod.Where(x => x.Id == guid).FirstOrDefaultAsync() ??
8.         throw new InvalidRequestException("The fishing method does not exist");
9.
10.    context.FishingMethod.Remove(methodToDelete);
11.    await context.SaveChangesAsync();
12. }
```

3.4.2 Special Features

3.4.2.1 Manual Location Syncing

When a user logs into Trace, they must select one of Talley’s site locations to sign into (e.g. Nelson, Motueka, Ashburton). Each location has its own database with its own data. This data is regularly synced between locations for most table (likely weekly), however for the fishing methods page has an additional button that manually triggers the syncing of fishing methods (see Figure 1). This button is only enabled when the user is signed into the Motueka location.

First, the list of fishing methods is retrieved from the Motueka database, which is the “master” database (see Figure 22, line 14). In the database, each site has a connection string and a name. Using these values, the method finds all valid sites it can connect to the database of (see Figure 22, lines 26-24). The rest of the method iterates through each of these databases, creating a connection, working out what changes need to be made to match with the mater database, and updating the fishing method items (see Figure 22, 26-89).

Figure 22: Sync fishing methods API method.

```

1. public async Task SyncFishingMethods()
2. {
3.     await using var dbContext =
4.         await _contextFactory.CreateDbContextAsyncWithDefaultConnection(_dbConnections);
5.
6.     // Must be in master plant to update
7.     if (_dbConnections.Default.ConnectionString != _dbConnections.Master.ConnectionString)
8.     {
9.         throw new InvalidRequestException(
10.            "You must be in the master branch to perform this action."
11.        );
12.    }
13.
14.    var masterFishingMethods = await dbContext.FishingMethod.ToListAsync();
15.
16.    var defaultPlant = await dbContext.Site.SingleAsync(o => o.IsDefaultPlant == true);
17.
18.    // Get all other sites (connection and name combination is different)
19.    var sites = await dbContext.Site
20.        .Where(s => !string.IsNullOrEmpty(s.dbConnection)
21.            && !string.IsNullOrEmpty(s.dbName)
22.            && !(s.dbConnection.Equals(defaultPlant.dbConnection)
23.                && s.dbName.Equals(defaultPlant.dbName))
24.        ).ToListAsync();
25.
26.    // Go through all other sites and update fishing methods
27.    foreach (var site in sites)
28.    {
29.        try
30.        {
31.            using (var remoteTransactionScope =
32.                new TransactionScope(
33.                    TransactionScopeOption.Suppress,
34.                    TransactionScopeAsyncFlowOption.Enabled
35.                )
36.            )
37.            {
38.                using var remoteDbContext = await _remoteContextFactory
39.                    .CreateDbContextAsyncWithConnectionString(// Connection);
40.

```

```
41.         var localFishingMethods = await remoteDbContext.FishingMethod.ToListAsync();
42.
43.         // Work out what needs to be added, updated, or deleted
44.         var existingMethodIds = localFishingMethods.Select(x => x.Id).ToHashSet();
45.         var updatedMethodIds = masterFishingMethods.Select(x => x.Id).ToHashSet();
46.
47.         var MethodsToAdd = masterFishingMethods
48.             .Where(updatedMethod => !existingMethodIds.Contains(updatedMethod.Id))
49.             .ToList();
50.
51.         var MethodsToDelete = localFishingMethods
52.             .Where(existingMethod => !updatedMethodIds.Contains(existingMethod.Id))
53.             .ToList();
54.
55.         var MethodsToUpdate = masterFishingMethods
56.             .Where(updatedMethod => existingMethodIds.Contains(updatedMethod.Id))
57.             .ToList();
58.
59.         // Add new methods
60.         foreach (var newMethod in MethodsToAdd)
61.         {
62.             remoteDbContext.FishingMethod.Add(newMethod);
63.         }
64.
65.         // Delete removed methods
66.         remoteDbContext.FishingMethod.RemoveRange(MethodsToDelete);
67.
68.         // Update same methods
69.         foreach (var updatedMethod in MethodsToUpdate)
70.         {
71.             var existingMethod = localFishingMethods
72.                 .First(x => x.Id == updatedMethod.Id);
73.
74.             if (!AreMethodsEqual(existingMethod, updatedMethod))
75.             {
76.                 existingMethod.FishingMethodName = updatedMethod.FishingMethodName;
77.                 existingMethod.IsActive = updatedMethod.IsActive;
78.                 existingMethod.MethodCode = updatedMethod.MethodCode;
79.             }
80.         }
81.         await remoteDbContext.SaveChangesAsync();
82.     }
83. }
84. catch (Exception)
85. {
86.     // Do nothing
87. }
88. }
89. }
```

3.4.2.2 Custom Read-Only Component

The need for this component originally stemmed from the need to have read-only text fields that could be automatically populated using React Hook Form but not allow the user to change the values. The first iteration to this solution was using the HTML disabled property, which initially met the requirements, however any disabled inputs would not have their values included in the form submission, which lead to issues when updating items. The next solution was to use MUI's *readOnly* property. This would prevent the user from editing the value, but the value would be submitted. A custom component was created to apply this property as well as custom styling for easy implementation in the UI (see the prompt code in Appendix A Prompt 1). The final hurdle was that some inputs are required to be read-only when editing but must be editable when creating a new item (else the value cannot be set). While using React's conditional

rendering via a ternary to render either a read-only text field or a normal text field would have worked, this would become messy and verbose in forms where this was required many times. Instead, creating a custom React component would allow for a much cleaner and dynamic UI.

The new read-only component takes a *readOnly* property, which can be controlled from the form (see Figure 23). Most forms are loaded with an *isCreate* boolean from the parent page, depending on whether an item is being updated or created. This value can be fed directly into any read-only components to easily return a read-only text field or a normal text field depending (see Figure 23). Due to the lack of online resources surrounding making custom components with a *styled* wrapper and custom input properties, AI was used to expand the original custom component with the *readOnly* property (see Appendix A Prompt 1).

Figure 23: Custom React read-only text input component.

```

1. import { styled } from "@mui/system";
2. import { TextFieldElement, TextFieldElementProps } from "react-hook-form-mui";
3.
4. interface ReadonlyTextFieldProps extends TextFieldElementProps {
5.   readOnly?: boolean;
6. }
7.
8. const StyledReadonlyTextField = styled(TextFieldElement)({
9.   "& .MuiInputBase-input": {
10.     color: "#888",
11.     cursor: "default",
12.   },
13. });
14.
15. const DynamicReadonlyTextField = ({readOnly = true, ...props}: ReadonlyTextFieldProps) => {
16.   if (readOnly) {
17.     return <StyledReadonlyTextField {...props} InputProps={{ readOnly: true }} />;
18.   }
19.   return <TextFieldElement {...props} />;
20. };
21.
22. export default DynamicReadonlyTextField;

```

3.4.2.3 Retrieving Image Names Through Network Folder

Each captain has an image of their name and an image of their signature. These images are stored on Talley's network drive, with the location path being stored in the captain's record in the SQL database. The list of these images is shown as dropdown options for the captain's page, so the user can set the correct image for the captain's name and signature.

To achieve this, the base network path for the captain images folder must first be retrieved from the database (this path is modified if in a testing environment) (see Figure 24, lines 6-17). The image names are then retrieved from all subfolders and returned to the client in a list when the page is loaded (see Figure 24, lines 19-40). The legacy application used many VB6 subroutines to handle the logic, including a verbose function to identify if the programme is being run in a testing environment by leveraging VB6's behaviours. Due to the verbosity and lack of knowledge in this area, an LLM was used to generate an explanation of the code (see Appendix A Prompt 2). With .NET's newer file and directory handling methods, these subroutines could be simplified into a much more concise method (Microsoft, n.d.b; Microsoft, n.d.c).

Figure 24: *Get skipper images API method.*

```
1. public async Task<IEnumerable<string>> GetSkipperImages()
2. {
3.     await using var context =
4.         await _contextFactory.CreateDbContextAsyncWithDefaultConnection(_dbConnections);
5.
6.     // Get network path
7.     string networkPath = await context.TraceSetting
8.         .Where(x => x.Setting == "CaptainImagePath")
9.         .Select(x => x.Value)
10.        .FirstOrDefaultAsync()
11.        ?? throw new Exception("Network path value not found");
12.
13.    // If in development mode
14.    if (Debugger.IsAttached)
15.    {
16.        networkPath = networkPath.Replace("// Path");
17.    }
18.
19.    // Get every file name
20.    List<string> files = [];
21.
22.    // Main folder
23.    string[] fileEntries = Directory.GetFiles(networkPath);
24.    foreach (string fileName in fileEntries)
25.    {
26.        files.Add(Path.GetFileName(fileName));
27.    }
28.
29.    // Subfolders
30.    string[] dirs = Directory.GetDirectories(networkPath, "p*", SearchOption.AllDirectories);
31.    foreach (string dir in dirs)
32.    {
33.        fileEntries = Directory.GetFiles(dir);
34.        foreach (string fileName in fileEntries)
35.        {
36.            files.Add(Path.GetFileName(fileName));
37.        }
38.    }
39.    return files;
40. }
```

3.4.2.4 Alphanumeric Comparer Helper Class

The need for a way to order items of a type based on an alphanumeric property arose from complications around the *metal detector preset* and *robot stack pattern* sections. Since the key value these objects hold is an alphanumeric string, numbers were being ordered incorrectly ([“1”, “10”, “2”] instead of [“1”, “2”, “10”]). As some values were letters, simply converting to an integer for the ordering wouldn’t work. Instead, a comparison code block was written to order all the values, whether number or letter, correctly.

Since this issue occurred across multiple places, the code block was extracted and expanded into a generic method what could take a list of any type and order based on the supplied property (see Figure 25). While this did significantly increase the complexity of the code to correctly get the type and property on each object, having this generic method available to all future pages is very beneficial.

Figure 25: *Alphanumeric comparer API helper class.*

```

1. public static class ComparerHelper
2. {
3.     /// <summary>
4.     /// Order a list of a specified type by a string alphanumeric property.
5.     /// Lists in numeric then alphabetic ascending order (1,2,10,A,B).
6.     /// </summary>
7.     /// <param name="array">Array of type T, with a string alphanumeric property ("A", "1",
8.     /// etc)</param>
9.     /// <returns>Ordered list (ascending numbers then ascending strings)</returns>
10.    public static T[] AlphanumericPropertyComparer<T>(T[] array, string orderProperty)
11.    {
12.        List<T> ints = [];
13.        List<T> strings = [];
14.
15.        for (int i = 0; i < array.Length; i++)
16.        {
17.            string value = array[i]
18.                .GetType()
19.                .GetProperty(orderProperty)
20.                .GetValue(array[i], null)
21.                .ToString();
22.
23.            if (int.TryParse(value, out int _))
24.            {
25.                ints.Add(array[i]);
26.            }
27.            else
28.            {
29.                strings.Add(array[i]);
30.            }
31.        }
32.
33.        // Order by value of orderProperty
34.        var orderedInts = ints.OrderBy(
35.            x => Convert.ToInt64(x.GetType().GetProperty(orderProperty).GetValue(x, null)));
36.
37.        var orderedStrings = strings.OrderBy(
38.            x => x.GetType().GetProperty(orderProperty).GetValue(x, null));
39.
40.        return [.. orderedInts, .. orderedStrings];
41.    }
42. }

```

3.4.2.5 Location Label Printing

Basic details about each location can be printed onto a label. Printing is handled through an external software called BarTender, which supplies intelligent label design and printing solutions for supply chain management (Software, n.d.). In Trace, from inside the editing page users can see a printing panel, which allows them to select the printer and number of labels to print (see Figure 26). Additionally, users can enable a ‘batch mode’, which allows them to select exactly which locations to print labels for (see Figure 26 and 27). Clicking the print button triggers a callback function in the parent page, sending the print request to the API. Here the details are converted into a BarTender request and sent to Bartender’s API, including appropriate error handling and validation (see Figure 28).

Figure 26: Location barcode printing UI panel.

Barcode Printing


FISHSHED	10.10.0.207
HALF SHELL	10.10.0.73
I.T OFFICE	10.20.1.169
IC CHURN ROOM	10.10.0.212
IC DRY GOODS UP	10.10.0.72
ICE PTR	10.10.0.12
IQF FISHSHED	10.10.0.205
M.C.S	10.10.2.253
MOT IC PAL	10.10.0.9
MOT IC PAL 2	10.10.0.12
MOT MARINATE PAL	10.10.0.202
MUS PTR	10.10.1.88
RACELOCAL	10.0.0.123
ROBOT MUS	10.10.0.202


☒ Batch Print

PRINT

Figure 27: Location batch mode UI panel.

Location Select For Printing

	ID	Plant	Warehouse	Position	Barco
<input type="checkbox"/>	1984	Motueka PH1	Bulk Store Main	J14C	L0101
<input type="checkbox"/>	1985	Motueka PH1	Bulk Store Main	J14D	L0101
<input type="checkbox"/>	1986	Motueka PH1	Bulk Store Main	J15A	L0101
<input type="checkbox"/>	1987	Motueka PH1	Bulk Store Main	J15B	L0101
<input checked="" type="checkbox"/>	1988	Motueka PH1	Bulk Store Main	J15C	L0101
<input checked="" type="checkbox"/>	1989	Motueka PH1	Bulk Store Main	J15D	L0101
<input type="checkbox"/>	1990	Motueka PH1	Bulk Store Main	J16A	L0101
<input type="checkbox"/>	1991	Motueka PH1	Bulk Store Main	J16B	L0101
<input type="checkbox"/>	1992	Motueka PH1	Bulk Store Main	J16C	L0101
<input type="checkbox"/>	1993	Motueka PH1	Bulk Store Main	J16D	L0101



2 selected

page 194 |< < > >|

Figure 28: Location label printing API method.

```
1. public async Task PrintLocationLabel(int[] locationIds, string windowsPrinterName,
2.   int numberToPrint)
3. {
4.     if (string.IsNullOrEmpty(windowsPrinterName))
5.     {
6.         throw new PrintFailedException("No printer specified.");
7.     }
8.
9.     if (locationIds.Length == 0)
10.    {
11.        throw new PrintFailedException("No locations specified.");
12.    }
13.
14.    if (numberToPrint < 1)
15.    {
16.        throw new PrintFailedException("Cannot print less than one copy.");
17.    }
18.
19.    await using var dbContext =
20.        await _contextFactory.CreateDbContextAsyncWithDefaultConnection(_dbConnections);
21.
22.    foreach (var id in locationIds)
23.    {
24.        var location = await dbContext.Location.SingleOrDefaultAsync(o => o.LocationId==id)
25.            ?? throw new PrintFailedException("Invalid location id.");
26.
27.        if (location.Barcode == null)
28.        {
29.            var plant = await dbContext.Site.SingleOrDefaultAsync(
30.                o => o.PlantDescription == location.PlantName
31.            ) ?? throw new PrintFailedException("Could not find site to generate barcode.");
32.
33.            location.Barcode = GenerateLocationBarcode(plant.PlantNumber, id);
34.            await dbContext.SaveChangesAsync();
35.        }
36.
37.        var labelRequest = new PrintLabelRequest()
38.        {
39.            Printer = windowsPrinterName,
40.            LocationBarcode = location.Barcode,
41.            LocationPlant = location.PlantName,
42.            LocationWarehouse = location.Warehouse,
43.            LocationPosition = location.Position,
44.            Label = Path.Combine(
45.                _options.LabelPath,
46.                _options.LocationLabelsPath,
47.                "Location - QR - 200x100.btw"
48.            ),
49.            NumberOfLabels = numberToPrint
50.        };
51.
52.        var result = await _printApiClient.PrintLabel(labelRequest);
53.
54.        if (!result.PrintedSuccessfully())
55.        {
56.            throw new PrintFailedException(
57.                $"Unable to print label for {location.Barcode}, stopping print job."
58.            );
59.        }
60.    }
61. }
```

3.5 Architectural Patterns

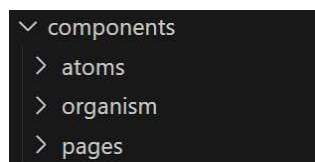
The system has utilised a few architectural patterns across its layers and technologies. React does not enforce an exact architecture, but instead provides a flexible component-based approach. The API uses a layered architecture, dividing it into interaction, application, and data layers. API tests are written using the AAA pattern for better structure and clarity.

3.5.1 React Client

While React is just a UI library and does not enforce its own architecture onto applications, leaving this decision up to the developer, it does operate with a component-based structure (Tutorials Point, n.d.a). Components are reusable, self-contained units of logic and UI, like pages, forms, and inputs, which form the building blocks of the application (Geeks for Geeks, 2025). Breaking down the application to smaller independent pieces make managing the application easier and allow reusability of UI and logic, reducing code duplication and speeding up development cycles (Geeks for Geeks, 2025; Teambits, 2024). Updates and bug fixes are localised to components, helping to identify and fix bugs, improving the system's overall maintainability and robustness (Geeks for Geeks, 2025; Teambits, 2024). Self-contained components give structure to the application, which can easily be scaled by adding new components (Geeks for Geeks, 2025; Teambits, 2024). This architecture also promotes collaboration as developers can work on different components simultaneously (Teambits, 2024). Component-based architecture does have its share of challenges however, especially with large-scale React applications, as state management can become complex, passing data between components can add overhead, and separating interdependent functionality can be difficult (Geeks for Geeks, 2025). This can inadvertently lead to a less clearly structured application, with maintenance issues, performances reductions and testing difficulties if not managed effectively (Geeks for Geeks, 2025).

The Trace React client implements component-based architecture by having separate components for each page, form, data context, and custom element. For example, the section for managing *boats* would use the following components: boat list page, boat editing page, boat editing form, boat data and boat data context for additional editing data, and possibly some custom element components like a read-only text field. The core components of the web client UI are organised based on a simplified version of the Atomic Design methodology (Frost, 2016) (see Figure 29). Atoms represent foundational building blocks like HTML elements and are the smallest level of components (Frost, 2016). Organisms are groupings of atoms and other organisms that form distinct sections of interfaces (Frost, 2016). Pages bring together atoms and organisms to create full pages that the user will see and interact with (Frost, 2016).

Figure 29: File structure of Trace's core component.



The web client also implements the container/presentation pattern, also known as the smart/dumb pattern (Hands on React, n.d.). The editing pages act as smart containers, which load and modify data via API calls and provide behaviour to buttons in the form (Hands on React, n.d.). These pages have minimal UI components except for calling the form with its data (Hands on React, n.d.). The editing forms are built as dumb presentation components, which receive the data and callbacks via props, and focus on how the UI looks (in this case the form layout and inputs) (Hands on React, n.d.). The container/presentation pattern promotes separation of concern in the structure of the application, with the ideas of improved code clarity, reusability, testability, and development efficiency (Patterns, n.d.; Naumov, 2020).

3.5.2 C# .NET API

While selecting or developing the architecture was not a component of this project, it is beneficial to recognise the architectural pattern used in a system to implement clean, quality code. The API layer of this project was implemented using a layered architecture pattern. The interaction layer serves as the interface between the API and the web client, managing incoming requests, handling authentication, and communicating responses (Catchpoint, n.d.). This layer includes the controllers and authorisation checking. Next is the application layer, with services that defines the core operations to handle requests, host business logic, and mapping of models (Catchpoint, n.d.). The data layer is handled within EF Core and the SQL Server database, and is responsible for storing, retrieving, and manipulating data (Catchpoint, n.d.).

3.5.3 API Tests

xUnit integration and unit tests for the API layer are structured using the AAA pattern. This is a common pattern that divides the code in a test method into three key sections: arrange, act, and assert (see Figures 4 and 5) (Microsoft, 2025d). The arrange section handles the initialisation of objects and setting of values that are required for the test (Microsoft, 2025d). The act section invokes the target method with the arranged parameters (Microsoft, 2025d). The assert section then verifies whether the called method behaved as expected or not (Microsoft, 2025d). The AAA pattern promotes better code organisation by enforcing the distinct separation of phases, as well as improving the clarity of tests by making them easier to understand (Zanini, 2025).

3.6 Visual Basic 6.0 in 2025

Despite the Microsoft Visual Basic 6.0 (VB6) language receiving its last update in 1998 (Abto Software, 2025), and losing development support in 2008 (Microsoft, 2024d), VB6 applications are still being used across healthcare, retail, finance, construction, and other industries (Abto Software, 2025). While VB6 does still run on the latest versions of Windows, there is no official method for installing the Visual Basic Integrated Development Environment (IDE) without

workarounds (Microsoft, 2024d), with Microsoft's stating themselves that they "strongly recommend that you replace your application with modern technology" (Microsoft, 2024d).

While development is still possible with VB6, there are many reasons why more modern technologies should be used instead. The termination of VB6's support means that there will be no more security updates, fixes, or new features, leaving accumulating security vulnerabilities, compatibility and stability issues, increasingly outdated features, and slower performance (Abto Software, 2025). Limited access to modern tools, frameworks, libraries and integrations restrict software development, leads to insufficient functionality, performance, and higher development times (Abto Software, 2025). Overall, migrating existing VB6 applications to modern technologies results in a much more secure, maintainable, and performant application with a more efficient development experience.

3.7 AI in Development

In accordance with academic integrity, artificial intelligence (AI) has been used sparingly and selectively during this project, with any uses during development fully referenced including the model and prompt used. In the professional industries however, use of AI is often much more relaxed or even encouraged due to the perception of time and manpower savings through automation, greater efficiency compared to humans, and possible reductions in the human workforce (Babashahi et al., 2024). In software development specifically, AI could reduce development time and costs through code generation, bug detection, optimising resources, and freeing developers for higher-level tasks (Ajiga et al., 2024; Sauvola et al., 2024). A 2025 research report by Google Cloud found that 98% of organisations are actively exploring the use of generative AI, with 39% already deploying it in production (Google Cloud, 2025). This shows just how rapid the adoption of AI has been in the industry and how much the development environment is changing. There are some concerns being expressed by around the ethics of using AI in the workplace, such as how to fairly and safely collect, use, and store data (Cebulla et al., 2022). Other concerns have been raised about how AI will affect job security, worker autonomy, and worker status, however with how recent the integration of AI into the workplace is, there is lacking research or evidence on the affects AI will have (Cebulla et al., 2022).

While no formal research or surveys of developers were conducted during this project, some insights were gathered through informal discussions and observations which is relevant to this project. AI use was not noticeably widespread in Talley's software development process, with only the occasional expression of interest expressed, predominantly using the integration of GitHub Copilot in Visual Studio (Microsoft, n.d.a). Most developers expressed a general lack of trust that it would generate high-quality code or even working code for non-trivial tasks, generally preferring to just write the code themselves so they knew it worked and how. One developer did express a higher interest in trialling LLM generated code, stating that he had replaced a section of code he had previously written with a much more optimised section of LLM generated code that was much cleaner. Another developer also used Copilot to identify improvements in my code and generate refactored code for several peer reviews during the project. Overall, the general sentiment from the development team was a lack in confidence of AI's ability to generate quality code, but small interest to test its use during development, and more confidence in using it as a feedback tool to improve code quality.

4 Analysis

4.1 System Value

This project has delivered a robust, high-quality, capable master data management system with a user-friendly interface. While the full migration is not complete, the purpose of this project was only to migrate a substantial portion of the RMS Admin, putting the development in a much better position to finish the migration and to help “push” the transition to Trace. The expectation when this project was proposed was to get 50-70% of the core features migrated, with the ideal burndown estimating 70% (see Figure 12). Out of the section list developed throughout the length of the project, 70% of section (also 70% of total story points) have been successfully migrated, which matches the high threshold of the expected results. Both Talley’s customer support representative (who manages RC testing) and software architect stated that very good progress had been made at the migration over the course of this project.

Due to Talley’s development processes and pipeline, every page and feature added throughout this project was peer reviewed by one or more developers, tested both independently and integrated with the current release, and has a wide coverage of both unit and integration tests. These all ensure that every page and feature is completed to a robust, professional, and fully functional standard with high code quality. Each page and functionality have been designed for clarity and user-friendliness, making Trace an effective and useful tool to manage Talley’s master data. With clarification of permissions, the completed pages should be ready for Talley’s staff to begin using them.

4.2 Plan Adherence

Due to the nature of this project where I was just working through tickets at my own pace instead of an end-to-end development process, I had a much less strict plan for the project. Essentially, my plan was to work through each page in the order given to me, or in the order of which pages I felt confident in my ability to make once I was creating my own tickets, and to complete as many as I could in the time frame.

This meant that the main issues I had were technical roadblocks where something wasn’t working as intended or a task was outside of my knowledge. With support from the other developers to help me debug problems, give advice, and guide me in the right direction, I was able to overcome all issues and continue with development. Sometimes this would cause a ticket to be put on hold for a couple of days, or even a few months in the case of the boats section, but with the flexibility of switching between tickets, I was able to spend my time on other work until I was able to revisit the problem.

Scheduling wise, I was able to stick closely to my original plan, however I lost occasional work hours to public holidays or extra classes which set me slightly behind (see Table 3). I also gradually lost a small amount of writing hours by either not being far enough into the project to be able to write much of the report, or by not quite being able to hit my ideal time goals around other work and study. As I approached the end of the project, I did end up reworking my

schedule to allow for more report writing earlier (see Table 3). This helped make sure I was on top of my report and helped break up the writing instead of having the last two weeks as pure report writing. I was still able to complete the vast majority of the work hours originally planned, only dropping a small portion to ensure that I had enough time to produce a high-quality report (see Table 3**Table 3**). Ideally, I should have worked slightly more in the first term to give me more leeway later in the project to catch up on any lost time and focus on the report. Note, most of our class time doubled as report writing time, making the report writing hours appear less.

Table 2: *Timeline of completed hours during the summer internship portion of the project*

Week	Time Completed (hours)
1	30
2	30
3	30
4	30
5	30
6	30
7	30
8	30
9	15
10	30
Total	285

Table 3: *Scheduled vs actual timeline for the academic portion of the project*

Week	Scheduled time (hours)			Actual Time (hours)		
	Work	Class/Writing	Total	Work	Class/Writing	Total
1	-	7	7	-	3.5	3.5
2	21.5	7	28.5	20	9.25	29.25
3	21.5	7	28.5	21.5	5.25	26.75
4	21.5	7	28.5	20.75	7.75	28.5
5	21.5	7	28.5	22.5	5.75	28.25
6	21.5	7	28.5	22	6.75	28.75
7	21.5	7	28.5	20.25	2.75	23
8	21.5	7	28.5	17.75	11.5	29.25
Break	30	4	34	29.75	3.25	33
Break	30	4	34	22.5	4.5	27
9	21.5	7	28.5	21	4.75	25.75
10	21.5	7	28.5	16.5	12.25	28.75
11	21.5	7	28.5	-	15.75	15.75
12	21.5	7	28.5	-	10.25	10.25
13	3.5	23	26.5	15.5	5.25	20.75
14	-	23	23	15.5	18.25	33.75
15	-	12	12	-	(poster)	(poster)
Total	300	150	450	265.5	126.75	392.25

Note: Some of the project time and work to develop the poster presentation will take place after

the completion of this report and cannot be included.

Due to the nature of this project where I was working on many different tickets and features, sometimes switching between over half a dozen branches in a day when sorting peer review or RC feedback, the development process was quite convoluted. In order to retain an accurate log of what I worked on, when I worked on it, and my thought process for solving more complex issues for both our assignment and for personal reference, I decided to write an extensive journal. This ended up taking up a lot more time and effort than it should have, which put a little strain on the time I had for other writing/work. It would have been beneficial to have scaled down the journal earlier in the progress to focus more on the important aspects of what significant work I achieved along with my reflections, without as many explanations for smaller details. This would have saved me significant time over the course of the project, allowing more time to focus on completing work hours and writing the report.

4.3 Development Process

While I have touched on agile methodologies before, it was fascinating to work in a larger-scale industry team and learn their development processes. I enjoyed working with the agile properties of DevOps as it gave me a lot of flexibility around my work, which was ideal for a modernisation project. Being able to work with the testing and support teams for both feedback and direction helped make the work I produced higher quality and more closely aligned with the user needs. This collaboration helped me learn much more about the overall system and how it is used, which lead to developing Trace with that knowledge in mind. As well as more effective development, the open collaboration and communication made the project more enjoyable. While I was not heavily involved with the automation and deployment, I can see that for a company like Talley's who manages their own internal systems, being able to automate these pipelines helps save a lot of time, making regular iterative updates much more achievable.

Historically, I've always favoured GitHub as a tool over Azure DevOps, however this project that completely changed for opinion. For managing a larger project with multiple people, the ease of having a wiki, repositories, ticket boards, backlog, pull request management, organisation management, pipeline automation all under one tool makes the process much easier. While GitHub and GitHub Projects do have many of these features, the way they are implemented in DevOps feels much more intuitive and easier to use. I really enjoyed the Azure DevOps environment and DevOps process.

4.4 Challenges

Overall, I believe that I had an effective process, demonstrated by the capable and quality application developed. The processes and experiences I have picked up on through the projects I have completed in my degree gave me a strong foundation of knowledge for picking up the DevOps process and having my own development process. There were however some challenges or areas that I could have improved in.

I am confident in my ability to work independently to tackle challenges. Throughout this project, I would always undertake my own research and try implementing solutions for any issues faced. By working through challenges myself, I can learn and grow my skills. I also know when it is better value of time to ask for help from a coworker than to waste time not making progress. One area I was trying to work on improving over the course of the project was being able to communicate with other developers exactly what I was trying to achieve, how I was trying to achieve it, and what the issue was, in a clear and efficient manner so they can best understand the issue. This can be quite a difficult task when working with a technical challenge in a part of the system the developer is not familiar with, however I think that I have improved my ability to do this.

The obvious change that would have significantly improved the efficiency of development would have been entering this project with more knowledge around the technologies being used. While I had a strong fundamental knowledge of React and C# .NET, I have never used technologies like EF Core and React Hook Forms, so having experience with these would have cut down on a lot of the time spent becoming familiar with them. This learning process was a valuable component of the project however, as it is where I gained a large portion of the skills and knowledge I have taken away.

One of the key challenges of this project was the lack of specificity in the pages to be migrated and the features they included. At the beginning of the project, tickets were created for the first selection of pages, but from then on, I was selecting which page I wanted to migrate next and creating the ticket myself. I have no inherent issue with creating my own tickets, as the flexibility to choose which page to migrate meant that I could pick the one I felt the most confident in my ability to make or choose how complex I wanted the next ticket to be. The primary issue was that I was not provided with a definitive list of which pages needed migration and which did not, and the rest of the developers and support staff weren't fully clear on it either. This is understandable as RMS Admin has dozens of pages and functionality that has been added over decades, some of which are used by one or two people, or used once per year, so it is hard for developers or support staff to keep track of exactly what is used and by who. This meant I had to combine the knowledge of different people on what they thought was and wasn't used, to make an approximate migration list. Even the software architect, my manager, gave me a rough list of items that included a page using a depreciated database table, which we then scrapped from migration. A vague and evolving work list makes it hard to estimate the total progress of the migration, and difficult to select which page to migrate next, especially later in the project when the obvious pages have already been completed. I would have found it very beneficial if before this project started, the IT team put together a definitive list of what is still used and needs migration, and what is no longer used, so everyone involved in the project was clear on exactly what was expected.

Another related issue I had was having incomplete tickets. While the first selection of tickets was provided, some included no details or just a list of the input fields for the form, which weren't always complete lists either. I did have access to RMS Admin which allowed me to use the existing pages as guides for creating the new pages, however this wouldn't tell me if there were any improvements to make or any unused features to remove, as the migrated pages weren't always the same. Many times, this would lead to RC feedback to add or change parts of pages that were missed or incorrect, adding more work for me and the testers. It would have been beneficial to have someone familiar with RMS Admin and its users creating the tickets with detailed information to better guide my development and reduce the work for both the

testers and me. Realistically, I could have also done a better job of fully exploring the legacy pages to ensure that I had created the new page as close as possible, as there were a couple of times that I missed features (e.g. an extra checkbox only shown when creating not editing a location) or implemented them slightly differently (e.g. as an editable text field not read-only).

In combination, as the migration work was the lowest priority work compared to bug fixes and new features for their existing systems, it often took much longer to get peer reviewed, testing, and RC feedback on my tickets, sometimes taking weeks. As each section was similar, I often reused code and adapted it to the new entity. This meant that by the time I received the feedback, I had often completed several more tickets reusing the same piece of code that the feedback replaced or improved. This resulted in both reviews and testers identifying and commenting on the same issue across multiple pages instead of once, and me having to apply the fix or change across many branches, resulting in much more combined work than if I had been able to immediately apply the feedback to my branch. It would have been beneficial to have feedback done more regularly and sooner on my branches, however I understand that each developer has their own work they are trying to prioritise, and that the influx of my tickets added a significant amount more work on top of the normal quantity of tickets.

I found that having large amounts of feedback to work through, combined with more complex pages later in the project, more missing/changing features from my initial page requirements, and receiving accumulated delayed feedback, meant that I had significantly less time to work through new tickets near the end of the project. This initially felt like I was being held back from my real work and making progress, but I have come to understand that this is just as important as the initial development of new tickets. A ticket is not truly completed until it is in the live release, often require more time and effort than just the initial development to make sure it has been implemented correctly and at a high standard.

One area I can work on which reduces the amount of feedback required and thus the amount of work for me to do, is to ensure that I am submitting the highest quality code in my pull requests as I can. While this is challenging when working with technologies I am less knowledgeable with, another method I implemented during the project was to create and expand upon a personal pull request (PR) checklist. Whenever I would get feedback from a peer review which I could implement across all pages, I would add it to a list which I would run through before submitting any future PRs. This helps to ensure that I am submitting completed quality work, saving time of other developers from having to find and comment on the same issues and my time having to fix them for every PR.

4.5 Alternative Approaches

As discussed previously (see section 3.1), there are many popular alternative technologies that could have been used to develop Trace. These include replacing EF Core with Dapper or React with Angular. Blazor would have also been a strong frontend technology, creating a unified C# .NET stack at the cost of being a newer tool with a smaller community, less support, and a slow initial render (Beres, 2025; Siddique, 2025).

Besides these obvious alternatives, there are many other possible approaches that could have been taken. One of the most common approaches to software development, especially for enterprise-level applications is Java stack. Having been around since the early 1990s, the Java

has accumulated a vast ecosystem of libraries and tools which facilitate rapid development (Salo, 2024). With high performance, scalability, strong security, and ease of maintenance, Java is suitable to almost any enterprise-level application (Salo, 2024). Web frameworks like JavaServer Pages and JavaServer Faces allow for creating dynamic and interactive web pages, while UI frameworks like JavaFX are popular for creating powerful, visually appealing user interfaces (Davis, 2024). While a Java-based stack is widely used in the industry, having come from primarily VB6 systems, Talley's team has limited knowledge and experience with Java, which would require a lot more learning, time, and cost to develop with in comparison to current C# .NET API.

Coming from the same .NET ecosystem, another alternative would have been to migrate from VB6 to VB.NET, the newer version of Visual Basic. VB.NET evolves upon VB6 bringing in many modern language features including complete object-orientated concepts, strongly typed variables, and a new runtime processor (ByteScout, 2016, Tutorials Point, n.d.b). While the code would not be directly compatible, it would be much closer and require less work to migrate, and be closer to the team's existing skillset. Microsoft has stated that they do not plan to continue developing VB.NET, so while the language is still in support, developing Trace with this language would not future-proof, making it no an ideal choice (Ramel, 2020).

Also sticking with the .NET ecosystem, C# offers UI frameworks with Windows Forms (WinForms) which hold similarities to the existing VB6 system through a related language, as well as drag-and-drop UI design (Chandran, 2023). WinForms is easy to learn, has rich controls, and solid performance, however it is often considered an aging technology with limited future development (Chandran, 2023). Like VB.NET, it does not offer ideal future-proofing, meaning it would likely require modernisation much sooner than selecting a more modern technology.

Using a technology like Blazor allows the development of client web applications with C#, which would unify the frontend and backend under one language (Diaz, 2024). A unified language simplifies development and knowledge required, which would align well with Talley's team's skillset (Diaz, 2024). Like React, Blazor uses component-based architecture to encourage reusability, modularity, and maintainability (Diaz, 2024). It offers high performance, and strong compatibility with the .NET ecosystem (Diaz, 2024). Despite being a new technology that may have some minor limitations, Blazor is a powerful tool and would have been well suited to this project (Diaz, 2024). Both React and Blazor have high capabilities for developing enterprise-level applications, and the decision on which to use ultimately concerns personal preference and skills of the team.

4.6 Insights

Throughout the undertaking of this project, I have been able to reinforce many of the skills I have learned throughout my degree in a real-world setting, as well as pick up new knowledge to complement them. I have a deeper understanding of React and C# .NET, and feel more confident in being able to apply them to future projects. I now have knowledge of new tools and technologies, including xUnit, EF Core, TypeScript, and Material UI, which I can use to enhance and support the development of future applications to create robust, professional, and scalable systems.

The experience of working in a real development team, being a part of their processes, meetings, and project management has been invaluable for preparing me for the start of my career. Having real work experience in the industry adds weight to back up my knowledge, which I believe will give me an advantage over those entering the industry who do not. I feel much more confident in my ability to now transition from study into the industry.

Over the course of the progress, I found that feedback from peer reviews, testing, and RC took up more time than I had expected it to. As a student, I am used to working on my own projects, where I have control over the scope and quality of code, and there are leniencies in the project compared to a live production project. From my experience now working in a real-world situation, I have learnt how important code reviews are to ensuring the quality and consistency of the system. Outside of their primary focus of programming, developers have other responsibilities to the project and their team, including putting the time into reviewing others code.

5 Future Development

While this project has been completed, the modernisation of RMS Admin in its entirety is still a work in progress. Most pages in RMS Admin that are used by staff have been migrated into Trace, leaving only a fraction left, as well as enhancements to the system over time. This remaining work will be completed by Talley's employed developers as users are transitioned across to the new system.

5.1 Short-Term

The immediate work to follow on from this project is the completion of the migration. All pages that are still used in RMS Admin and have not been migrated yet will have to be developed. While this is required for a full transition to Trace, some users who only work with pages that have already been migrated can begin converting to Trace. By staggering the user transition with a staged migration, any bugs or issues discovered will be spread out over time (D'Entrone, 2023). This approach helps to contain risks, as well as helping the development team not become overwhelmed while debugging and deploying any required fixes (D'Entrone, 2023).

One important improvement will be finalising the new roles and permission system. In RMS Admin, permissions are controlled through an attribute-based access control, where each user has specified permissions for all actions (SailPoint, 2023). Trace however is moving to a role-based access control, where users are assigned roles (e.g. product admin, order admin), which give them permissions to perform actions (SailPoint, 2023). The ability to access pages and perform actions will then be locked behind related roles. This system will have to be implemented at least at a basic level before users transition to using the client, otherwise they will be able to read, edit, and delete data that they should not be able to. Once a fundamental access is determined, this can be easily expanded upon or changed by simply adding new roles and giving them to the selected accounts.

5.2 Mid-Term

As users begin using Trace, they will likely encounter bugs or ways the system could behave better. This will require iterative improvements from the development team to fix any discovered bugs and continue to improve the system. Improvements could range from new features, such as the ability to manually synchronise data between locations on more pages, or adapting the pages to be closer to the business rules, such as switching fields to be read-only, modifying required fields, or implementing stricter validation.

Some pages and tools in RMS Admin are used by a few users at most, or very infrequently, making it hard to know exact what is used and what isn't. While the developers and support staff have a general idea of what is used, it is likely that over the near future there will be functionality requested that was not migrated. As these are discovered, they will have to be

added to the system. This should help reduce the overall bloat from RMS Admin, as any unused features should be left behind, improving the user experience.

5.3 Long-Term

Once all the required features are migrated, Trace will move into a polishing phase, then finally a maintenance phase. During maintenance, developers will be focussing primarily on fixing any bugs discovered by users or adapting the system to meet changing business and user requirements. At this stage, it may be worth examining the system to see if there are any other improvements to be made to keep the system modern and relevant. One likely change would be to replace any existing service methods that use Dapper with EF Core for a unified system. Another suggestion would be to perform a cleaning and normalisation project on the database to reduce bloat from data duplication and unused columns. A well-maintained database will make future development easier and improve the data quality.

6 Conclusion

This project successfully migrated the majority of Talley's legacy master data management system (RMS Admin) into the new system, Trace. This means that Trace is functional to accurately manage most of the master data entities in Talley's business domain. The initial estimation was the migration of 50-70% of the legacy system, with the project completing exactly 70%. The conclusion of this successful project has put Talley's in a strong position to finish modernising the system and transition the userbase over from the legacy application.

Compared to VB6, which has poses security, performance, compatibility, and maintainability issues due to being outdated and out of support, the modern technologies used to construct Trace result in a high-quality system. React promotes a much more developer-friendly environment, with high code reusability, prebuilt components from libraries like Material UI, performant virtual DOM, and easy scalability, making it a powerful tool for both developing and maintaining a web client like Trace's. Using C# .NET for the API has expansive support through Microsoft's ecosystem, with a large community, many integrations, and easy scalability, likewise making it a dominant technology for API layers. In addition, using an ORM like Entity Framework Core further simplifies the interaction between API and database, which made the development of Trace more efficient and easier.

Three key challenges were faced during this project. The first challenge highlighted by the project was how a lack of knowledge impacts the development of a system. While obvious in concept, the effect of this can clearly be shown by the exponential completion rate of tickets over the first half of the project (see Figure 12). The burndown chart demonstrates how progress started off slowly, falling behind the ideal burndown, then caught up and overtook the ideal burndown as the understanding of the system and its technologies grew. This also illustrates the skills both gained and developed throughout the project, demonstrated by the accelerated development speed in the middle of the project. The second key challenge faced was a combination of vague ticket requirements and an undefined feature list. For the beginning of the project, tickets were supplied by other developers to guide the development, however for the rest of the development the tickets were self-made. These supplied tickets often had vague or no details attached, making it difficult to ensure that the created pages achieved the desired result without requiring a number of feedback cycles to reach completion. Self-made tickets had to be guided fully by exploration of the existing legacy system, which meant no or minimal input on what parts weren't used or could be improved. The issues of self-made tickets were magnified by an undefined feature list, due to the developers not knowing exactly which parts of the system were still used and which weren't. Once the obvious pages had been migrated, this made it challenging to identify which pages to migrate next. This combined challenge was combatted through extensive research of the existing system to guide development, as well as compiling knowledge of system use from the other developers, testers, and support staff. The final key challenge was posed due to the lower priority nature of this migration in relation to the other developer's work of adding feature and fixing bugs in Talley's current systems. This resulted in longer feedback cycles, meaning that code was reused across more branches in the meantime. This causes more work to apply feedback across all required branches and more work from the other developers identifying these issues in pull requests, resulting in an overall less efficient development process. Working with the team to aim for a faster turnaround would

have saved time for everyone as well as speeding up development.

In addition to substantial progress in the modernisation of the legacy master data management system for Talley's, this project has provided many opportunities for reinforcing and acquiring both knowledge and skills across React, C# .NET, EF Core, and other technologies. As well as technical ability, exposure to a real-world, industry workplace has provided valuable experience on how development processes and methodologies like agile and DevOps are implemented and used to manage the development of multiple large-scale applications. The feedback cycle of peer reviews, branch testing, and release candidate testing was supplied a lot of insight into how to construct robust, high-quality code in a professional environment, as well as demonstrating the other responsibilities of developers.

Despite the completion of this project, Talley's will continue the migration of RMS Admin into Trace, expanding its functionality. Eventually RMS Admin will be fully replaced and the userbase transitioned over to Trace. This gives Talley's a much more maintainable, robust, performant, and development-friendly system to manage their master data for years to come.

7 References

- Abto Software. (2025, February 20). *Is Visual Basic still used in 2025?* <https://www.abtosoftware.com/blog/is-visual-basic-still-used-in-2024>
- Ajiga, D., Okeleke, P. A., Folorunsho, S. O., & Ezeigweneme, C. (2024). Enhancing software development practices with AI insights in hightech companies. *Computer Science & IT Research Journal*, 5(8), 1897-1919. <https://doi.org/10.51594/csitrj.v5i8.1450>
- Angular Minds. (2025, March 3). *Top reasons to use React for enterprise app development in 2025.* <https://www.angularminds.com/blog/react-for-enterprise-app-development>
- Atlassian. (n.d.). *DevOps.* <https://www.atlassian.com/devops>
- AWS. (n.d.). *What is DevOps?* <https://aws.amazon.com/devops/what-is-devops/>
- Babashahi, L., Barbosa, C. E., Lima, Y., Lyra, A., Salazar, H., Argôlo, M., de Almeida, M. A., & de Souza, J. M. (2024). AI in the workplace: A systematic review of skill transformation in the industry. *Administrative Sciences*, 14(6). <https://doi.org/10.3390/admsci14060127>
- Beres, J. (2025, February 18). <https://www.infragistics.com/blogs/blazor-vs-react>
- Biri, A. (2024, August 24). *Exploring the best languages to develop REST API.* BSuperior System. <https://bsuperiorsystem.com/blog/best-languages-to-develop-rest-api/>
- Bisbal, J., Lawless, D., Wu, B., Grimson, J., & Wade, V. (1997, January). *A survey of research into legacy system migration.* Trinity College Dublin. <https://publications.scss.tcd.ie/tech-reports/reports.97/TCD-CS-1997-01.pdf>
- Bogard, J. (n.d.). AutoMapper. <https://automapper.org>
- Bourne, V. (2022). *Size of COBOL.* Commissioned by Micro Focus. https://www.vansonbourne.com/case-studies/size-of-cobol/?utm_source=chatgpt.com
- ByteScout. (2016). *Migrating VB 6 code to Microsoft VB.NET.* <https://bytescout.com/blog/2016/01/differences-between-vb-6-and-vbnet.html>
- Catchpoint. (n.d.). *API architecture patterns and best practices.* <https://www.catchpoint.com/api-monitoring-tools/api-architecture>
- Cebulla, A., Szpak, Z., Howell, C., Knight, G., & Hussain, S. (2022). Applying ethics to AI in the workplace: the design of a scorecard for Australian workplace health and safety. *AI & Society*, 38, 919-935. <https://doi.org/10.1007/s00146-022-01460-9>
- Chandran, R. (2023, November 6). *Advantages and Disadvantages of Winforms?* Software Developer India. <https://www.software-developer-india.com/advantages-and-disadvantages-of-winforms/>
- Das, J. (2024, April 5). *What is hybrid agile methodology and how to implement?* Orangescrum. <https://blog.orangescrum.com/what-is-hybrid-agile-methodology-and-how-to-implement>

- Davis, E. (2024, May 15). *Exploring Java in Frontend Development*. BizCoder.
<https://bizcoder.com/java-frontend/>
- D'Entrone, F. (2023, November 15). *Migration center*. Large content migration strategies: Big bang vs. phased approach. <https://migration-center.com/blog/large-content-migration-strategies-big-bang-vs-phased-approach>
- Diaz, I. (2024, July 3). *Using Blazor in Enterprise Applications*. Apiumhab.
<https://apiumhub.com/tech-blog-barcelona/using-blazor-enterprise-applications/>
- Duvvur, V. (2023, July). Legacy systems in the age of big data: Modernization strategies. *International Journal of Science and Research (IJSR)*, 12(7), 2281-2283.
<https://doi.org/10.21275/SR24517161400>
- Ehsan, A., Abuhaliqa, M. A., Catal, C., & Mishra, D. (2022). RESTful API testing methodologies: Rationale, challenges, and solution directions. *Applied Sciences*, 12(9), 4369.
<https://doi.org/10.3390/app12094369>
- ErikEJ. (2025, March 26). *EF Core Power Tools (Version 2.6.961)*. Visual Studio Marketplace.
<https://marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools>
- Fanelli, T. C., Simons, S. C., & Banerjee, S. (2016). A systematic framework for modernizing legacy. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, (pp. 678-682). Osaka, Japan.
<https://doi.org/10.1109/SANER.2016.40>
- Figuerola, A., Rollo, S., & Murthy, S. (2017). *A brief comparison of two enterprise-class RDBMSs*. Department of Computer Science. Western Connecticut State University.
<https://doi.org/10.48550/arXiv.1710.08023>
- Forsberg, M. M. (2022). *An evaluation of .NET Object-Relational Mappers in relational databases: Entity Framework Core and Dapper*. Department of Computing Science. Umeå University. <https://umu.diva-portal.org/smash/record.jsf?pid=diva2%3A1687513&dswid=4572>
- Frost, B. (2016). Atomic Design. <https://atomicdesign.bradfrost.com/chapter-2/>
- Fryč, D. (2024). *Web Components UI Library*. Department of Software Engineering. (Master's thesis) Czech Technical University in Prague.
<https://dspace.cvut.cz/bitstream/handle/10467/114666/F8-DP-2024-Fryc-Dominik-thesis.pdf>
- Fürnweiger, A. (2017). *A legacy system migration and software evolution*. Technische Universität Wien. <https://doi.org/10.34726/hss.2017.26665>
- Geeks for Geeks. (2025, March 12). *React component based architecture*.
<https://www.geeksforgeeks.org/react-component-based-architecture>
- Google Cloud. (2025). *State of AI infrastructure*.
https://services.google.com/fh/files/misc/google_cloud_state_of_ai_infra_report.pdf
- Greenough, C., & Worth, D. (2006). *The transformation of legacy software: Some tools and a process (Version 3)*.

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=329ed0e1334e8f428f743f0c542aaa622d4acc45>

Greplová, M. (2023). *Dealing with legacy systems and software redesign*. Masaryk University.

Hakeem, A. (2024, February 29). *Best programming languages to develop REST API*. TooUet.
<https://blog.tooljet.ai/best-programming-languages-for-rest-api-development>

Hands on React. (n.d.). *Component architecture*. <https://handsonreact.com/docs/component-architecture>

Hikmawati, S., Santosa, P. I., & Hidayah, I. (2021). Improving data quality and data governance using master data management: A review. *International Journal of Innovative Technology and Exploring (IJITEE)*, 5(3).
<https://journal.ugm.ac.id/ijitee/article/viewFile/66307/32204>

IBM. (2021, October 8). *What is infrastructure as code (IaC)?*
<https://www.ibm.com/think/topics/infrastructure-as-code>

Ihnatovich, D. (2024, September 27). *Debouncing and Throttling in React: What's the Difference and How to Implement Them*. Medium.
<https://medium.com/@ignatovich.dm/debouncing-and-throttling-in-react-whats-the-difference-and-how-to-implement-them-0a500b649235>

Innocent, A. (2025, March 11). *C#, Java, Golang, or Python: Which is the best language for API development?* Apidog. <https://apidog.com/blog/best-languages-for-building-apis>

Khadka, R., Batlajery, B. V., Saeidi, A. M., Jansen, S., & Hage, J. (2014). *How do professionals perceive legacy systems and software modernization?* Utrecht University.
<https://doi.org/10.1145/2568225.2568318>

Khadra, F. [. (2025). *react-toastify (version 11.0.5)*. npm.
<https://www.npmjs.com/package/react-toastify>

Kharnagy. (2016, September 7). DevOps toolchain [SVG image]. Wikimedia Commons.
<https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg>

Kumari, R. (2024, June 7). *What is peer review in software testing?* Testsigma.
<https://testsigma.com/blog/peer-review-in-software-testing>

Laoyan, S. (2025, February 20). *What is Agile methodology? (A beginner's guide)*. Asana.
<https://asana.com/resources/agile-methodology>

Loser, C., Legner, C., & Gizanis, D. (2004). *Master data management for collaborative service processes*.
https://www.researchgate.net/publication/44938150_Master_Data_Management_for_Collaborative_Service_Processes

Loshin, D. (2010). *Master data management*. Morgan Kaufmann.

Marko, K. (2016, May 24). *Best programming languages for enterprise development*. TheServerSide. <https://www.theserverside.com/feature/Best-programming-languages-for-enterprise-development>

Material UI. (2025). <https://mui.com>

Material UI. (n.d.a). *Button*. <https://mui.com/material-ui/react-button>

Material UI. (n.d.b). *Table*. <https://mui.com/material-ui/react-table>

Microsoft. (2023a, March 28). *Creating and configuring a model*.
<https://learn.microsoft.com/en-us/ef/core/modeling>

Microsoft. (2023b, August 25). *Getting started with EF Core*. <https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app>

Microsoft. (2024a, October 4). *About work items and work item types*.
<https://learn.microsoft.com/en-us/azure/devops/boards/work-items/about-work-items>

Microsoft. (2024b, November 12). *Entity Framework Core*. <https://learn.microsoft.com/en-us/ef/core>

Microsoft. (2024c, October 5). *What is SQL Server?* <https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server>

Microsoft. (2024d, December 19). *Support statement for Visual Basic 6.0 on Windows*.
<https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/visual-basic-6-support-policy>

Microsoft. (2025a). *C#*. <https://dotnet.microsoft.com/en-us/languages/csharp>

Microsoft. (2025b). *Microsoft SQL Server*. <https://www.microsoft.com/en-us/sql-server>

Microsoft. (2025c). *TypeScript is JavaScript with syntax for types*.
<https://www.typescriptlang.org>

Microsoft. (2025d, February 28). *Unit test basics*. <https://learn.microsoft.com/en-us/visualstudio/test/unit-test-basics>

Microsoft. (2025e, April 23). *What is Azure DevOps?* <https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops>

Microsoft. (n.d.a). *Copilot free in Visual Studio 2022*.
<https://visualstudio.microsoft.com/github-copilot>

Microsoft. (n.d.b). *Directory.GetDirectories method*. <https://learn.microsoft.com/en-us/dotnet/api/system.io.directory.getdirectories>

Microsoft. (n.d.c). *Directory.GetFiles method*. <https://learn.microsoft.com/en-us/dotnet/api/system.io.directory.getfiles>

Mishra, A. (2020). Legacy system modernization: effective strategies and best practices. *International Journal of Leading Research Publication (IJLRP)*, 1(3).
<https://www.ijlrp.com/papers/2020/8/1245.pdf>

Naumov, A. (2020, January 16). *Separation of concerns in software design*. NalexN.
<https://nalexN.github.io/separation-of-concerns>

Ogunwole, O., Onukwulu, E. C., Joel, M. O., Adaga, E. M., & Ibeh, A. I. (2023). Modernizing legacy systems: A scalable approach to next-generation data. *International Journal of Multidisciplinary Research and Growth Evaluation*, 04(01), 901-909.
<https://doi.org/10.54660/IJMRGE.2023.4.1.901-909>

- OpenAI. (2024). ChatGPT (GPT-4-turbo, March 2024 version). <https://chat.openai.com/chat>
- Oracle. (2021, May 14). *What is business intelligence?*
<https://www.oracle.com/nz/database/what-is-data-management>
- Pansara, R. (2021). Master data management challenges. *International Journal of Computer Science and Mobile Computing (IJCSMS)*, 10(10), 47-49.
<https://doi.org/10.47760/ijcsmc.2021.v10i10.008>
- Patadiya, J. (2023, December 18). *What makes ReactJS the perfect choice for enterprise application development in 2025?* Radix. <https://radixweb.com/blog/reactjs-for-enterprise-app-development>
- Patterns. (n.d.). *Container/presentational pattern*.
<https://www.patterns.dev/react/presentational-container-pattern>
- PractiTest. (2024, March 1). *Unit test vs integration test*. <https://www.practitest.com/resource-center/article/unit-test-vs-integration-test>
- Pradhan, D. (2024, October 5). *Top 10 best programming languages for enterprise application development*. CredibleSoft. <https://crediblesoft.com/top-10-programming-languages-for-enterprise-application-development>
- Pujara, P. (2024, July 12). *Why ReactJS is a game changer for enterprise app development*. Dev. <https://dev.to/prashantpujara/why-reactjs-is-a-game-changer-for-enterprise-app-development-3jh1>
- Radigan, D. (n.d.). *An agile guide to scrum meetings*. Atlassian.
<https://www.atlassian.com/agile/scrum/ceremonies>
- Ramel, D. (2020, March 12). *Microsoft: 'We Do Not Plan to Evolve Visual Basic as a Language'*. Visual Studio magazine. <https://visualstudiomagazine.com/articles/2020/03/12/vb-in-net-5.aspx>
- React. (n.d.a). React (version 19.1). <https://react.dev>
- React. (n.d.b). *UseCallback*. React (version 19.1).
<https://react.dev/reference/react/useCallback>
- Rout, P. (n.d.). *Fluent API in Entity Framework Core*. Dot Net tutorials.
<https://dotnettutorials.net/lesson/fluent-api-in-entity-framework-core>
- SailPoint. (2023, March 10). *Types of access control systems*. <https://www.sailpoint.com/en-nz/identity-library/what-are-the-different-types-of-access-control-systems>
- Salo, L. (2024, April 4). *4 Benefits of Java for enterprise application development*. Vaadin.
<https://vaadin.com/blog/java-for-enterprise-applications>
- Salonen, S. (2023). *Evaluation of UI component libraries in React development*. Department of Information Technology. Tampere University.
- Sauvola, J., Tarkoma, S., Klemettinen, M., Rieki, J., & Doermann, D. (2024). Future of software development with generative AI. *Automated Software Engineering*, 31(26).
<https://doi.org/https://doi.org/10.1007/s10515-024-00426-z>

- Seacord, R. C., Comella-Dorda, S., Lewis, G., Place, P., & Plakosh, D. (2001). *Legacy system modernization strategies*. Carnegie Mellon University.
https://insights.sei.cmu.edu/documents/663/2001_005_001_13895.pdf
- Siddique, R. (2025, January 23). *Blazor vs React: Choosing the right framework for your project*. eLuminous. <https://eluminoustechnologies.com/blog/blazor-vs-react>
- Sivagnana Ganesan, A., & Chithralekha, T. (2017). A comparative review of migration of legacy systems. *International Journal of Engineering Research & Technology (IJERT)*, 6(2), 484-488. <https://www.ijert.org/research/a-comparative-review-of-migration-of-legacy-systems-IJERTV6IS020319.pdf>
- Software, S. (n.d.). BarTender. <https://www.bartendersoftware.com>
- Spec India. (2024, March 13). *MySQL vs MSSQL: What to choose for enterprise applications?* <https://www.spec-india.com/blog/mysql-vs-mssql>
- Stack Exchange Inc. (n.d.). Stack Overflow. <https://stackoverflow.com>
- Talley's. (2025-a). *Homepage*. Retrieved March 16, 2025, from <https://www.talleys.co.nz>
- Talley's. (2025-b). *Our brands*. Retrieved March 16, 2025, from <https://www.talleys.co.nz/about-us/our-brands>
- Talley's. (2025-c). *Regional roles*. Retrieved March 16, 2025, from <https://www.talleys.co.nz/careers/locations>
- Teambits, B. (2024, February 14). *A comprehensive guide to component-based architecture*. LinkedIn. <https://www.linkedin.com/pulse/comprehensive-guide-component-based-architecture-brandon-opere-k2ite>
- Teamhub. (2024, February 9). *Understanding the significance of release candidate in software development*. <https://teamhub.com/blog/understanding-the-significance-of-release-candidate-in-software-development>
- TechSteps. (n.d.). *The top programming languages in demand in NZ: A guide for aspiring developers*. Retrieved May 8, 2025, from <https://www.techstep.nz/topprogramminglanguages>
- Tutorials Point. (n.d.a). *ReactJS - Architecture*. https://www.tutorialspoint.com/reactjs/reactjs_architecture.htm
- Tutorials Point. (n.d.b). *VB.Net - Overview*. https://www.tutorialspoint.com/vb.net/vb.net_overview.htm
- Vige, W. (2025, May 17). *Story points: Estimation guide for user stories in Agile*. Asana. <https://asana.com/resources/story-points>
- Wolfart, D., Domingos, D. C., Assunção, W. K., Schmeing, E., Paza, D. d., Silva, I. F., & Villaca, G. L. (2021). Modernizing legacy systems with microservices: A roadmap. *EASE '21: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. Trondheim, Norway. <https://doi.org/10.1145/3463274.3463334>
- xUnit.net. (n.d.). *About xUnit.net*. <https://xunit.net>

Yaba, M. S. (2023). *Performance comparison between Entity Framework Core 6 and Dapper*. Tomas Bata University in Zlín.

https://digilib.k.utb.cz/bitstream/handle/10563/53922/yaba_2023_dp.pdf

Zanini, A. (2025, January 17). *The arrange, act, and assert (AAA) pattern in unit test automation*.

Semaphore. <https://semaphore.io/blog/aaa-pattern-test-automation>

ZZZ Projects. (2024a, October 17). *Welcome to Learn Dapper*. Learn Dapper.

<https://www.learndapper.com>

ZZZ Projects. (2024b, August 30). *What is Entity Framework Core?* Learn Entity Framework Core.

<https://www.learnentityframeworkcore.com>

Appendix A: AI Prompts for Development

The following prompts were provided to generative LLMs in the cases where other resources and current knowledge were not sufficient to achieve a solution.

Prompt 1: Custom Material UI Component

Submitted to ChatGPT, GPT-4-turbo, March 2024 version (OpenAI, 2024).

Prompt:

I have the following Material UI custom styled component. Modify this component to take a new custom prop 'readOnly', which should apply the custom styling and readOnly InputProps if true, else returning a standard text field.

Custom component: "

```
import { styled } from '@mui/system';
import { TextFieldElement, TextFieldElementProps } from 'react-hook-form-mui';
interface ReadonlyTextFieldProps extends TextFieldElementProps {}
const StyledReadonlyTextField = styled(TextFieldElement)({
  '& .MuiInputBase-input': {
    color: '#888',
    cursor: 'default',
  },
});
const ReadonlyTextField = (props: ReadonlyTextFieldProps) => {
  return <StyledReadonlyTextField {...props} InputProps={{ readOnly: true }} />;
};
export default ReadonlyTextField;"
```

Prompt 2: VB6 Function Explanation

Submitted to ChatGPT, GPT-4-turbo, March 2024 version (OpenAI, 2024).

Prompt:

Explain "Public Function InIDE(Optional ByRef bool As Boolean = True) As Boolean If bool Then Debug.Assert Not InIDE(InIDE) Else bool = True End Function"