OSPRI Internship

Adrian C. Lebita
Victoria University of Wellington, Masters of Software Development
Wellington, New Zealand
Lebitaadr@myvuw.ac.nz

***Abstract*—OSPRI is primarily responsible for animal traceability and disease management of livestock in New Zealand. It runs three programs, in particular MyOSPRI a cloud-hosted event-driven microservice Web API(Application Interface). MyOSPRI's primary function is to allow livestock owners to complete Animal Status Declarations(ASD) electronically through the MyOSPRI Portal. Prior to MyOSPRI, ASDs were completed on paper or by calling the OSPRI Contact Centre. Third party integration support is being developed for users who do not want to opt-in to the MyOSPRI Portal. This support is called Partner Integration or PAPI. The PAPI API is a layer built on top of the core services of MyOSPRI. It adopts the same technologies that MyOSPRI has been built on utilizing event-driven microservice architecture. Using this architecture allows the API to remain scalable and maintainable into the future.**

I. Introduction

As a requirement for completing a Masters in Software Development, I was to complete work experience relevant to the software industry. I had been given an internship opportunity at OSPRI, a company that is responsible for monitoring animal movements and livestock disease nationwide.

During my 3-months at OSPRI, I was tasked with backend developer duties for their upcoming third party integration support for Animal Status Declaration(ASD) management of the MyOSPRI program which has an expected release in July, 2022.

The goals I had set out to achieve during my internship were:

- Deploy microservice
- Use Test Driven Design for Code Implementation
- Learn JetBrains Rider IDE (Integrated Developer Environment) shortcut keys.

This report details my learning while at OSPRI including the company, the MyOSPRI project, system architecture, software delivery and reflection of learnings.

II. OSPRI

OSPRI (Operational Solutions for Primary Industries) is a not-for-profit disease management organisation for the cattle, lamb and deer industries of New Zealand. It is funded by Dairy NZ, Deer Industry New Zealand and Beef & Lamb New Zealand.

The main goal of OSPRI is to provide animal traceability and disease management services for New Zealand's animal export industry - an industry that requires transparency on farming practices and stock traceability.

Importers have strict regulations and guidelines that must be satisfied for successful trade. Assurance is needed of the products being exported; that they have been produced from industry-best practices, were free from contamination/disease and have been produced with great control and attention to product safety.

Failure to provide product integrity greatly affects market confidence resulting in less demand and can lead to adverse economic conditions as animal-related goods (meat, cheese, milk, butter and so on) account for nearly 45% of New Zealand's export trade.

OSPRI currently runs three programs for disease management and animal tracing.

1) NAIT: The NAIT (National Animal Identification Tracing) programme is an online portal where every owner of stock must register their animals for contact and tracing, whether for personal or commercial purposes. This information is also used by MPI (Ministry of Primary Industries) for disease management.

2) TB Free: A programme with its primary vision is to see complete eradication of TB in New Zealand by 2055. Services offered in the TB Free programme include disposing of infected stock, on-farm testing, animal quarantine, lab sampling and spatial data.

3) MyOSPRI: An online web service for managing ASD documents which contains key information about an animal or group of animals such as feeding times, movements and disease control. Prior to MyOSPRI, all ASD documents were completed on paper or by calling the OSPRI Contact Centre.

III. Software System

A. MyOSPRI Web API

The MyOSPRI program is delivered through an online web application that is developed internally by the Technology Solutions team. It is an event-driven, cloud-hosted, microservices application. The Technology Solutions team employ full Agile practices for development and has DevOps team for cloud-support, CI/CD (Continuous Integration/Continuous Delivery), support and health monitoring.

Main technologies implemented include Microsoft's Dotnet framework and Azure Cloud Hosting services, Angular JS for frontend, Docker for containerization and Kubernetes for microservices management.

B. Event Driven Architecture

MyOSPRI is based on event-driven architecture. Events are broadcasted to an event hosting service (or event bus) whenever a change in state has occurred, typically when data is updated.

An event-drive system consists of event producers and event consumers. Together they manage and maintain the state of the API. Event consumers are service-agnostic; that is when they listen on the event hosting service, they do not care from where, or which service produced the event. If the event matches the consumer's identifiers, the consumer will respond to the event.

Event producers raise events coupled with a signature and data. This enables the transfer of information between services that work independent of another system's state.

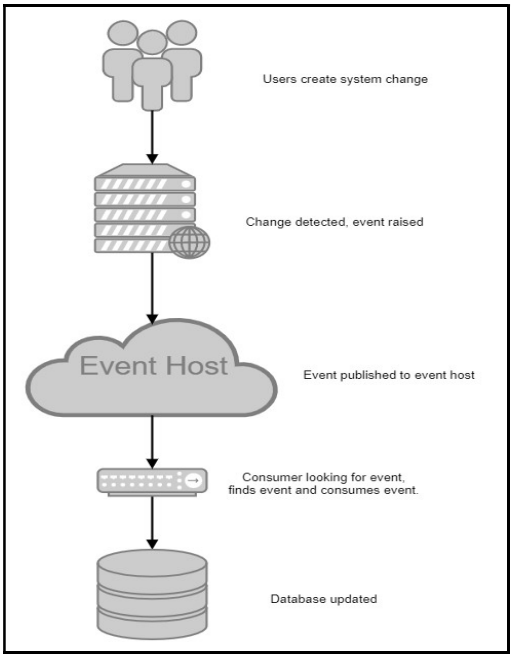

Fig. 1. Simplified sequence of event driven architecture to manage system state.

The benefit of event-driven architecture is that it provides a flexible system that can respond to real-time data and system states. Since events and consumers are service-agnostic and stateless, it naturally leads to a microservice architecture system.

C. *Microservice Architecture*

Microservice architecture is an approach to building a large application as small services. These services work independently of each other. The key benefit of implementing microservices is the ability to have higher up-time than traditional, monolithic applications.

For instance, in monolithic applications, a small update requires the entire application to be down for maintenance. With a microservice approach, only the service that requires the patch or fix will be offline, but other functionalities of the service are still available for use.

A great illustration of microservices is an online shopping web API. Imagine a customer can view, select and purchase products, and each of these actions is a service on its own. If for example, the purchasing service needed to be updated, the customer is still able to view and select products. This is a great advantage for complex applications that have many functionalities such as MyOSPRI.

Microservices also take full advantage of dependency injection patterns. Since microservices are essentially separate systems, dependency injection services can be specific. For example an authorization dependency may not be needed for a service that is not concerned with permissions. The ability to optimize dependencies increase performance, scalability and maintainability of the overall system.

Like events and consumers, microservices are also service agnostic. Despite being part of a larger API, internally microservices are unaware of the state of other services. For the microservices to work together, event-driven architecture are used to communicate information between each service.

Microservices are usually containerized in an isolated environment and is managed using an orchestration tool that monitors the health and performance of each service. MyOSPRI uses Docker for containerization and Kubernetes for service orchestration. The main benefit of containerizing these services is the ability to monitor with clarity the performance of each service and ease of scalability.

With the combined architectures of event driven architecture and microservices, the MyOSPRI API is a highly flexible and scalable application.

D. *Software Design*

MyOSPRI approaches software design using Domain Driven Design (DDD). Domain Driven Design is a way to order code from a top-down view of business logic. It encourages separation of concerns based on purpose, function and data modelling. This separation of technical complexity is called layers or logic artifacts.

As an example in Figure 2, there are nine logic artifacts within the Animals domain.

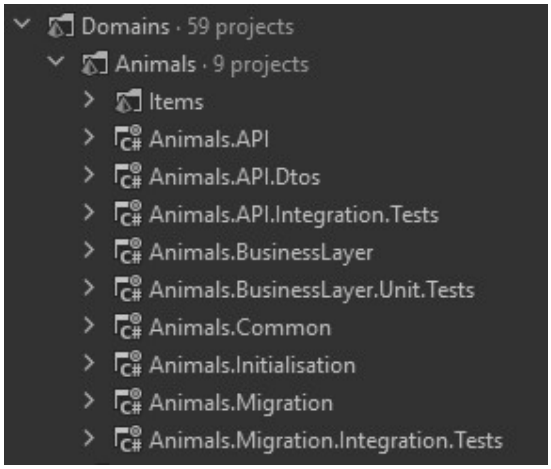

Fig. 2. Animals domain has nine logic artifacts of separation for business logic and data modelling.

Within each of the layers are models and class files that should not be propagated to other layers of the domain.

The purposes for each layer:

1) Animals.API: The application of the service. In here are simple startup files and software configurations for when

the software run. The point of entry into this service is in this layer.

2) Animals.API.Dtos: Dtos stand for Data To Object models. In simpler terms, the shape of data is stored here. For example, there is a SpeciesDTO file and within that class file it has properties such as 'Name', 'Id' and 'SpeciesType'.

3) Animals.API.Integration.Tests: Integration test cases are run and stored in this layer.

4) Animals.API.BusinessLayer: This is the business logic of the services. Files to do with DTO manipulation such as command and queries and object mapping are stored here.

5) Animals.BusinessLayer.Unit.Tests: Unit tests for business logic.

6) Animals.Common: Common resources used across all layers such as events, database and entities.

7) Animals.Initialisation: Services that are initialised at the same time the API layer is called.

8) Animals.Migration: A layer for simpling sepearting out migration models for the databases.

9) Animals.Migration.Integration.Tests: Tests cases to ensure data migration to database is successful.

Each domain may have its own set of business logic, entity models, database system and software configuration but they follow the same separation pattern constrained by Domain Driven Design. The MyOspri web API has five domains (effectively microservices), together they are referred to as 'Core'. Refer Figure 3 for simplified architecture of MyOSPRI.

For each domain, code files are further organized using Command and Query Responsibility Segregation (CQRS) pattern. The CQRS pattern approaches code segregation between reading and writing data. Using this approach, code implementation remains short and simple, and chaining multiple queries to databases can be done using basic object orientated principles. This makes complex logic simpler to understand and maintain.

E. Challenges with Current System

While the MyOSPRI API is flexible and maintainable, it does require a reasonably seasoned architect to guide and instruct developers, especially for complex domains, fuzzy business logic, entity relationships and integration with cloud hosting services. The only approach when encountered with such a complex system is to build a smaller version of the API in one's own time.

IV. PROJECT

A. Partner Integration

The MyOSPRI programme provides one option for using the API service to manage Animal Status Declaration forms by using the MyOSPRI portal; a frontend service developed in-house. An alternative option being developed is providing functionality for third-party users that may want to manage ASDs without the MyOSPRI portal. This project is called Partner Integration or PAPI and is scheduled to be released in July 2022.

B. Project Requirements

PAPI is a third-party API where authorized users can call specific endpoints to perform specific tasks relating to ASD management. The endpoints are hosted on SwaggerHub and displays the routes, parameters, schemas and expected HTTP

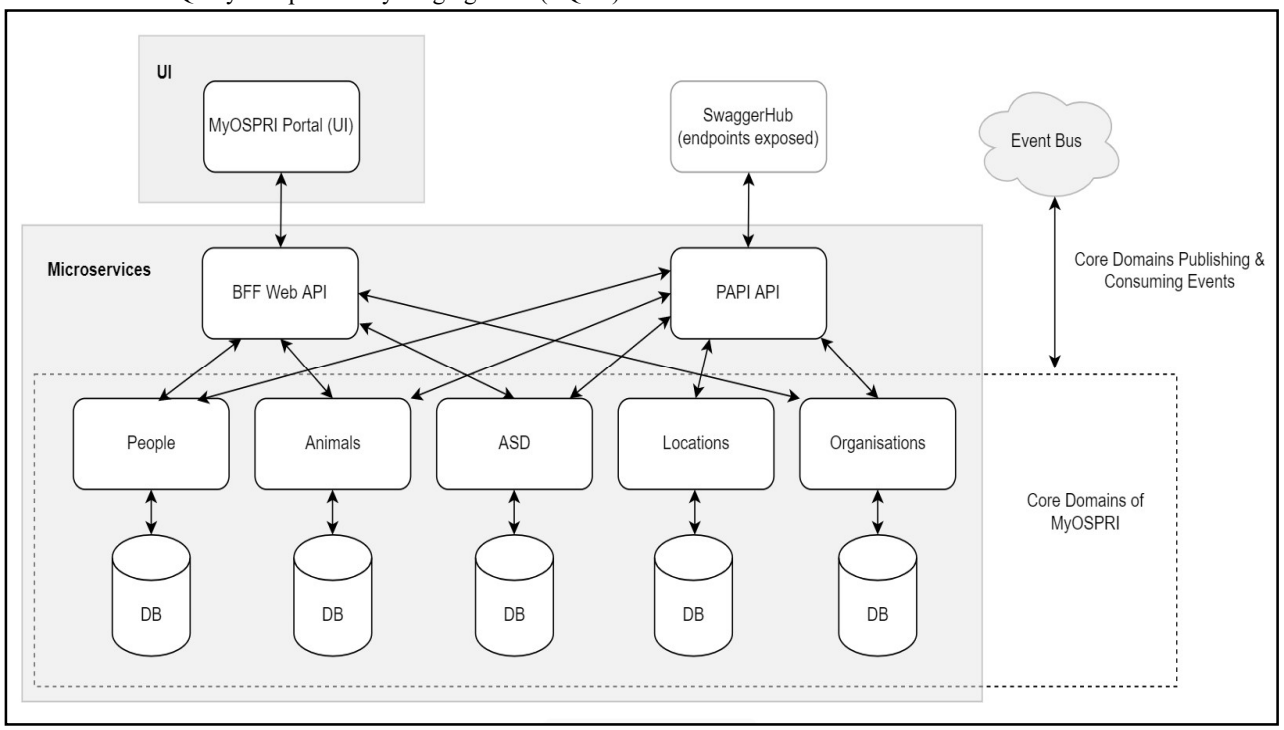

Fig. 3. Simple overview of microservice architecture implemented in MyOSPRI. Both PAPI and BFF Web API use core domains to read and write data.

requests and responses. Refer Figure 4.

There are five major features in PAPI, each handling a different aspect of ASD management. These features have been considered minimum viable product for PAPI to allow third-party users to perform minimum ASD functionality. These features are:

1) Farm to Meat Processor ASD: Allows Meat Processor users to manage ASDs and set statuses such 'reject', 'hold' or 'complete'.

2) Manage Locations: Users can manage their locations, whether they are opted in to receive ASDs and the Farm to Farm ASD: Farm users can send, fetch and update ASDs.

3) Manage ASD: Options for users to select animal transporters opted-in for ASD management, view regulatory questions and supplementary questions relating to ASDs.

4) Partner Foundation: Enable technology features, such as authorization, permissions and roles.

Note: Brevity used due to business sensitivity.

Each of these features are broken down into stories that then go through refinement. Stories are given a Scope, Acceptance Criteria and a Fibonacci rating based on complexity and duration. After a story has been refined, it is ready for development work.

C. Delivery Strategy

PAPI is scheduled to be code complete by close of business 10th of June 2022, and expected to be released in July 2022 to a small number of key users for feedback and performance.

The team uses a mix of Kanban and Scrum Agile practices to deliver work. Sprints are timeboxed fortnightly starting Wednesday and finishing on Tuesday. Refinements occur every Monday and Thursday to ensure that the sprint backlog contains enough work for developers and QA testers.

Since the development teams operate Kanban, sprint velocity isn't as important compared to other traditional Agile practices. So long as stories are being refined and there is not a considerable build up in any of the board columns, then development is an on-going activity.

The two-week timebox is to ensure that the team at some point come together to discuss retrospectives, share learnings and improvements.

OSPRI is committed to enhancing and improving the Agile methodology and even employs a full-time Agile coach to ensure Agile practices are adhered to.

D. Software Delivery

1) Integration Testing: A story is considered code complete when the there are passing integration tests. Integration tests are broad and test the system's components. As such creating and running these tests can be cumbersome. Unlike unit tests, which test a single code block in isolation, integration tests cover end point usage, HTTP request/response models, validation, mapping and data models. Completing local end-to-end tests also give opportunity to rectify any software build errors that may arise. After successful integration tests, developers run end-to-end tests locally on their machine. Services are containerized, event services are imitated and local databases are populated with mock data. Once setup, endpoints are then tested using an API testing program such as Postman. Developers call the endpoint and check that the system's behavior is correct for different inputs. Once finished, the code is made available for QA Testers to perform additional tests.

2) QA Testing: QA testing involves pseudo Black-Box Testing using only endpoint information provided in SwaggerHub.. Their goal is to verify that the software behaviour matches the Story's Acceptance Criteria. Testing is done on a range of inputs correct and incorrect to ensure proper responses. Any unexpected behavior encountered, a bug is raised and the story is given back to the developer.

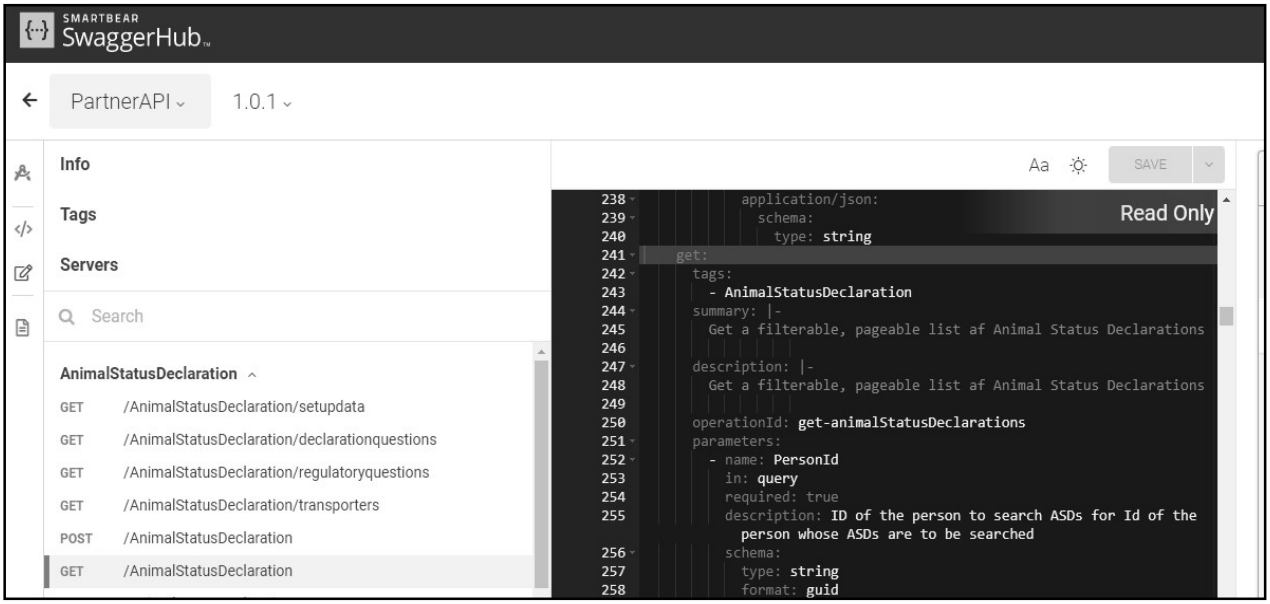

Fig. 4. Snapshot of SwaggerHub publicly exposing PAPI endpoints for third-party users.

3) SIT: When all stories of a feature are completed (developed and QA tested), the feature is put through several different environments before releasing to production. In OSPRI, these phases are System Integration Test (SIT), Staging and Preview. In SIT, testers check that the application meets the technical, functional and business requirements of the story. Typical tests include black-box testing of single or multiple features.

4) Staging: After SIT, the feature is brought into Staging. The staging environment is close as identical as can be to the production environment, including data, databases, computer resources, cloud services and any other services used in production. In this environment testers check to ensure there are no issues with deployment on cloud infrastructure and performance issues with software builds. Essentially ensuring that deploying the new feature will not adversely affect the current live environment.

5) Preview: Preview is where User Acceptance Testing (UAT) is performed. Users not involved with code development are given the opportunity to trial the new features ensuring it meets the specific needs of the User. Like Staging, this is done in a near identical environment of production.

6) Production: Finally after UAT has been verified and no issues identified with deployment, the feature is released into the production environment for live use.

7) Lifecycle Maintenance & Beyond: After releasing to Production, the Web API is monitored using Dev Ops tools for resource, maintenance and health checks. Tools that are used to monitor health of containerized applications and hosting services are Loki and Helm Charts.

With the MyOSPRI web API employing a microservice architecture and dependency injection pattern, maintainability is much easier for the lifecycle of the API and is easily scalable for emerging technologies.

Using Agile methodologies, development teams can rapidly deliver software that can go through rigorous testing checks. In fact, sometimes the time spent for a feature or story in SIT or Preview far outweighs the duration it was taken to develop.

V. Reflection

A. Goals & Learnings

The goals set out at the start of the internship were to challenge myself to get into good developer habits and gain fundamental knowledge in microservice architecture. While I did not achieve these goals in their entirety – I was able to complete a significant portion of each goal.

In the first couple of weeks, I was able to use Docker to containerize applications and run it locally. When developing code, I adopted Test driven development principles not for unit tests but for developing integration tests. By the end of the internship, I had mastered key IDE shortcuts that increased productivity and mental cadence.

Outside of these goals, majority of my learning at OSPRI has been split between four core areas.

1) Web API development: validation principles, mapping data, controller/route/endpoint implementation, commands and queries.

2) Learning software and technology: Such as Postman, MSSQL Server, Redis Insight, Auth0, Docker, SwaggerHub and Entity Core Framework.

3) Testing: Setting, creating and running integration tests and performing end-to-end testing.

4) ADO (Azure Dev Ops) Workflow: Learning how a feature is broken down into smaller stories. Then going through the different stages from ddevelopment through to production.

Despite my work experience and exposure of technologies at OSPRI, I believe that the most important attribute a developer can possess is genuine interest in technology. Interest in technology will drive self-learning and exploration, a key trait that is needed in a rapidly changing environment. Combine this trait with great communication; with proper guidance any developer could be great at delivering readable and maintainable software.

B. Relevance of Education

The Masters of Software Development course offered by Victoria University of Wellington has been extremely relevant to my work experience at OSPRI. The following specific modules; Web API, Agile practices, Industry Project and Object Orientated Principles has given me a stable foundation to transition into the workforce.

Although I have not experienced any mobile development, the Android module could be taught better. A great improvement would be to first develop a basic Android application rather than showing the Android Studio IDE.

A small but very important module (maybe half a day lesson before end of Week 3 in SWEN501) is understanding the debugging tool. In the workplace, it was simply assumed that I knew how to run debugger, read stack traces and threads. After completing my first story – about four-weeks in, I understood that the debugger tool was used comprehensively for TDD development.

Early exposure to the debugging tools allows developers to see execution flow, how data is stored and where in the code it's breaking. This would encourage comfortability around exception and error handling and promote problem-solving abilities early on in the course.

C. Covid-19

Covid-19 had negligible impact during my time at OSPRI. I was already working from home due to Omicron outbreak in mid-2021 during the Master's course; a safe and productive work environment was already prepared.

On the occasions I did go into the office I practiced safe distancing, maintained personal hygiene and wore masks in public spaces.

By the end of the internship, OSPRI had encouraged all employees to return to office as the first choice of work operations. This is not enforced giving employees the flexibility to do a combination of working in the office and from home.

VI. Conclusion

OSPRI employs a range of technology solutions to deliver the MyOSPRI Web API. While it can be complicated to setup, event-driven microservice architecture allows the API to remain scalable and maintainable into the future.

While majority of my time had been spent doing backend work, it would be nice to have gained some experience with development in the frontend space and working with the DevOps team.

Overall, OSPRI has been am invaluable experience and has taught me fundamental aspects of Web API development.

Acknowledgement

Big thank you to J. Rush for his encouragement to apply at OSPRI for an internship despite the job description sounding intimidating. Without his support I would've never applied.

References

[1] M. Edge, B, Kavalali, "NAIT Reivew: Final Report On The Recommendations", OSPRI, Wellington, New Zealand, Mar. 29, 2018. Accessed: May 05, 2022. [Online]. Available: https://www.beehive.govt.nz/sites/default/files/2018-04/NAIT%20Review%20Final%20Report.pdf#:~:text=NAIT%20provides%20this%20system%20for%20New%20Zealand%20and,the%20premises%20in%20which%20the%20livestock%20is%20held.

[2] "The TBfree programme | OSPRI", Ospri.co.nz, 2022. [Online]. Available: https://www.ospri.co.nz/our-programmes/the-tbfree-programme/. [Accessed: 01- Jun- 2022].

[3] N. Number, "MyNZBN", New Zealand Business Number, 2022. [Online]. Available: https://www.nzbn.govt.nz/mynzbn/nzbndetails/9429030199415/?sw=. [Accessed: 03- Jun- 2022].

[4] "New Zealand's Top 10 Exports 2021", Worldstopexports.com, 2022. [Online]. Available: https://www.worldstopexports.com/new-zealands-top-10-exports/. [Accessed: 03- Jun- 2022].

[5] "New Zealand Exports - May 2022 Data - 1951-2021 Historical - June Forecast - Calendar", Tradingeconomics.com, 2022. [Online]. Available: https://tradingeconomics.com/new-zealand/exports#:~:text=New%20Zealand%20Exports%20Exports%20in%20New%20Zealand%20advanced,cheese%20%2830%25%29%3B%20and%20meat%20and%20edible%20offal%20%2816%25%29. [Accessed: 03- Jun- 2022].

[6] "New Zealand Top Commodity Imports & Exports: A World Leader In Concentrated Milk Exports - Commodity.com", Commodity.com, 2022. [Online]. Available: https://commodity.com/data/new-zealand/. [Accessed: 03- Jun- 2022].

[7] Ospri.co.nz, 2022. [Online]. Available: https://www.ospri.co.nz/assets/Documents/ASD-Form.pdf#:~:text=The%20purpose%20of%20the%20Animal%20Status%20Declaration%20%28ASD%29,de%20mining%20expo%20eligibili%20y%20and%20certi%EF%AC%81%20cation. [Accessed: 03- Jun- 2022].

[8] "Microservices architecture", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture. [Accessed: 03- Jun- 2022].

[9] "Designing a DDD-oriented microservice", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice. [Accessed: 31- May- 2022].

[10] "Introduction to Containers and Docker", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/. [Accessed: 31- May- 2022].

[11] "Applying simplified CQRS and DDD patterns in a microservice", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/apply-simplified-microservice-cqrs-ddd-patterns. [Accessed: 31- May- 2022].

[12] "The Four Levels of Software Testing | Segue Technologies", Segue Technologies, 2022. [Online]. Available: https://www.seguetech.com/the-four-levels-of-software-testing/. [Accessed: 01- Jun- 2022].

[13] "Service-to-service communication", Docs.microsoft.com, 2022. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/service-to-service-communication. [Accessed: 01- Jun- 2022].

[14] R. Martin, Clean code. A handbook of agile software craftsmanship. Boston: Pearson Education, 2009.

[15] E. Evans and M. Fowler, Domain-driven Design: Tackling Complexity in the Heart of Software, 1st ed. Boston: Addison-Wesley, 2003.

[16] S. Newman, Monolith to Microservices: Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith, 1st ed. Sebastopol: Publisher O'Reilly Media, 2019.

[17] A. Bellemare, Building Event-Driven Microservices, 1st ed. Sebastapol: O'Reilly Media, 2020.