**Project Overview:**

**Domain:** Insurance

**Project Name:** LINQ Core Application (Sub-project of Marsh)

**Technical Details:** Java, Spring Boot, MYSQL, Angular.

**Summary:**

The LINQ Core Application is a business insurance platform designed to simplify the management of business insurance needs for companies. It provides a digital platform that streamlines processes like obtaining insurance quotes, managing policies, submitting claims, and monitoring compliance with insurance requirements. By offering a central solution for managing risk portfolios, the application enhances decision-making, simplifies data access, and improves interaction with insurance offerings for employees, customers, and contractors.

* Key stakeholders included the client (Archipology), business owners, product owners, development teams, and end users like insurance agents and policyholders.
* My role as a backend developer, to enhance the existing Business Insurance system based on the user stories assigned during sprint planning. I focused on improving specific modules like policy management and claims processing, fixing bugs, and integrating new features as required by the business.

**Challenges:**

1. This project involves complex calculations for premiums, policy coverage, claims adjudication and regulatory compliance. Implementing this logic internally led to performance issues or incorrect outputs if not well-optimized.

* Use Spring Boot's transaction management features to ensure that business-critical transactions, such as policy issuance or claims settlement, are executed atomically. Write unit tests and integration tests to validate the correctness of all business logic. SonarQube can be used to enforce good coding practices and prevent technical debt.

2. Maintaining code quality is essential in a long-term project, especially in this project, because it is a single application where business logic is concentrated.

* Integrate SonarQube into my CI/CD pipeline to continuously review the codebase. SonarQube helps catch issues like code smells, duplicated code, low test coverage, and potential security vulnerabilities.

3. Coordinating deployment, code quality checks, and test automation in a seamless CI/CD process can be complex.

* Used R2D2 orchestration to automate the build, test, and deployment processes. Set up automated workflows where new code is automatically tested, analyzed by SonarQube, and deployed after approval.

4. Ensuring smooth, secure, and fast communication between the Angular front end and Spring Boot back end, especially for data-heavy processes like claims or policy management.

* Implement RESTful APIs in Spring Boot with proper pagination and filtering to avoid overloading the backend. Use lazy loading in Angular for better performance by only loading the necessary components and data when needed. Optimize the Angular HTTP Client for handling large data responses efficiently (e.g., compressing JSON responses).

**Outcomes:**

This project is expected to deliver a highly secure, efficient, and scalable system that significantly improves the way insurance policies and claims are managed. The automation of manual processes, high code quality, and CI/CD capabilities contribute to long-term sustainability, allowing the business to grow and evolve with confidence.

**Learning Experience:**

Working with both Spring Boot on the backend and Angular on the frontend provides hands-on experience with full-stack development. I learnt how to build RESTful APIs in Spring Boot and integrate them with a responsive, user-friendly Angular front end.