**FACULTY OF SCIENCE, ENGINEERING AND COMPUTING**

**School of *Computer Science & Mathematics***

**BSc DEGREE**

**IN**

**Computer Science – Software Engineer**

**PROJECT PROPOSAL**

**Name:** M.W.M.M. ZAHEER

**ID Number:** E171743

**Project Title:** Retail Management System for Ruby Hardware

**Project Type**: Build

**Date**:

**Supervisor:** Mr. Vikum Jayasundara

KU London Logo

Did you discuss and agree the viability of your project idea with your supervisor? Yes

Did you submit a draft of your proposal to your supervisor? Yes

Did you receive feedback from your supervisor on any submitted draft? Yes

# Abstract

Ruby Hardware sells a wide range of products to customers in retail and wholesale sectors, including hardware tools, construction materials, home improvement products, and automotive parts. Using paper-based systems, the store manually maintains sales, inventory stock, and customer records. This approach results in inefficiencies and difficulties in maintaining accurate and up-to-date records.

Separate registers are used to record product, supplier, and sales information, which causes stock mismanagement, billing errors, time-consuming record-keeping, and security risks due to handling sensitive data. Furthermore, tracking purchase history and generating reports is challenging, requiring manually referencing physical records.

To overcome these challenges, a comprehensive Retail Management System will be developed to streamline Ruby Hardware's business operations. This web-based system will replace the existing manual processes and improve efficiency in sales, inventory management, purchasing management, customer management, employee management, and billing.

The system will be developed using modern technologies, the front end will be built with Next.js, along with UI improvements using Tailwind CSS and shadCn/UI components. The back end will be developed using Laravel, with PostgreSQL as the database for managing and storing data. API testing will be conducted using Postman.

Key features of the system include a point-of-sale (POS) interface for sales transactions, integrated inventory management for real-time stock updates, and purchase management to handle supplier and stock details. Additionally, it includes customer and employee management modules to maintain detailed records, billing and invoicing capabilities for automated receipt generation, and a dashboard that provides a centralized view of key business metrics such as sales statistics and stock levels.

With a role-based login system, the Retail Management System will ensure secure access for different user roles with appropriate permissions. The system will also generate detailed reports and analytics to provide valuable insights into business performance.

Contents

[Abstract i](#_Toc199611644)

[List of Figures/Tables iv](#_Toc199611645)

[Figures iv](#_Toc199611646)

[Tables iv](#_Toc199611647)

[1. Introduction & Background 1](#_Toc199611648)

[1.1. Introduction 1](#_Toc199611649)

[1.2. Background and Motivation 1](#_Toc199611650)

[1.3. Problem in brief 2](#_Toc199611651)

[2. Aim & Objectives 3](#_Toc199611652)

[2.1. Aim 3](#_Toc199611653)

[2.2. Objectives 3](#_Toc199611654)

[3. Technologies & Resources 4](#_Toc199611655)

[3.1. Technologies Used 4](#_Toc199611656)

[3.1.1. Technology Stack Overview 4](#_Toc199611657)

[3.1.2. Detail Description of Each Technology 5](#_Toc199611658)

[3.2. Resource Requirement 7](#_Toc199611659)

[4. Methodology & Work plan 11](#_Toc199611660)

[4.1. Methodology 11](#_Toc199611661)

[4.2. Major Phases of the Project 12](#_Toc199611662)

[4.3. Feasibility Study 18](#_Toc199611663)

[4.5. Major Milestones & Deliverables 19](#_Toc199611664)

[4.5. Contingency Plan 21](#_Toc199611665)

[4.6. Gantt Chart 22](#_Toc199611666)

[5. Proposed Solution 23](#_Toc199611667)

[5.1 Ethics 23](#_Toc199611668)

[5.2 Data Protection 24](#_Toc199611669)

[5.3 Safety 25](#_Toc199611670)

[5.4 Suggested Starting Point 26](#_Toc199611671)

[6. Discussion*.* 28](#_Toc199611672)

[6.1. Legal Issues 28](#_Toc199611673)

[6.2. Ethical Issues 29](#_Toc199611674)

[6.3. Societal Issues 30](#_Toc199611675)

[6.4. Security Issues 31](#_Toc199611676)

[7. References 33](#_Toc199611677)

[Appendices 35](#_Toc199611678)

# List of Figures/Tables

## Figures

[Figure 1: Next.js 4](#_Toc199611679)

[Figure 2: Laravel 4](#_Toc199611680)

[Figure 3: PostgreSQL 5](#_Toc199611681)

[Figure 4: Tailwind CSS 5](#_Toc199611682)

[Figure 5: ShadCn 5](#_Toc199611683)

[Figure 6: Agile (geeksforgeeks, 2025) 11](#_Toc199611684)

[Figure 7: Contingency Planning 21](#_Toc199611685)

[Figure 8: Gantt Chart 22](#_Toc199611686)

## Tables

[Table 1: Milestones Timeline Table 20](#_Toc199611687)

[Table 2: Appendix A - System Key Functional Modules 35](#_Toc199611688)

[Table 3: Appendix B - Database Schema Design 36](#_Toc199611689)

[Table 4: Appendix C – System Users Roles 37](#_Toc199611690)

# Introduction & Background

## Introduction

Ruby Hardware operates as a retail and wholesale supplier of various products, including hardware tools, construction materials, home improvement goods and automotive parts. Currently, Ruby Hardware relies on manual paper-based systems to manage sales, inventory, suppliers and customer data. This outdated approach has led to inefficiencies such as frequent errors, stock mismanagement and delayed reporting, affecting overall productivity and customer satisfaction.

This project aims to develop a web-based Retail Management System that automates and integrates key business functions such as sales, inventory, purchasing, customer and employee management and billing. This system will improve accuracy, reduce operational delays and enhance data security by providing real-time access to critical business information. This Project scope includes designing and implementing a user-friendly platform with role-based access, reporting features and automated billing to meet Ruby Hardware's operational needs.

## Background and Motivation

Effective sales and inventory management is essential for business growth and long-term profitability in today's fast-paced retail industry (Sommerville, 2016). However, many small and medium-sized retail businesses still rely on manual record-keeping methods, which are often error-prone and inefficient (Goyal & Gupta, 2020). With the increasing demand for automation and real-time data access, it has become clear that modernizing business operations is key to reducing costs and improving customer satisfaction (Gartner, 2023).

While a range of retail management tools exist-from basic spreadsheets to full-scale ERP systems these solutions are either too generic or too expensive for smaller businesses like Ruby Hardware. Manual processes also carry out the risk of lost records, weak data security, and slow reporting, making it difficult for business owners to make timely, informed decisions (Raj & Singh, 2021).

These challenges inspired this project, focusing on building a customized retail management system specifically for Ruby Hardware. The system will use modern web technologies to offer an affordable, scalable, and secure platform that automates sales through a user-friendly POS interface, tracks inventory in real-time, manages supplier and purchase records, and simplifies billing. It will also support management by offering insightful dashboards and reports that can be customized as needed.

By embracing digital transformation, the proposed system aims to improve Ruby Hardware's overall efficiency, empower staff with better tools, and provide smoother customer experience. Features like automated processes and secure, role-based access will lead to better decision-making and more effective resource use.

## Problem in brief

Ruby Hardware's reliance on manual paper-based systems for managing sales, inventory and customer records results in inefficiency, frequent errors, and delays in information retrieval. Separate physical registers for different types of records cause stock mismanagement, billing inaccuracies and data security concerns. These challenges hamper the store's ability to maintain accurate and timely business information, ultimately affecting operational effectiveness and profitability.

Addressing these issues is critical to ensuring Ruby Hardware can keep pace with market demands, maintain customer satisfaction, and make informed decisions based on reliable data. Therefore, developing a web-based Retail Management System that automates and integrates business functions is essential to improving record accuracy, reducing administrative workload, and enhancing overall business performance.

# Aim & Objectives

## 2.1. Aim

This project aims to develop a Retail Management System to automate and streamline Ruby Hardware's retail business operations.

## 2.2. Objectives

* To critically review the retail management problem domain, focusing on challenges in manual operations.
* To carry out a comprehensive study of available technologies that can be used to build retail management systems.
* To design and develop a fully functional Retail Management System tailored to the needs of Ruby Hardware.
* To evaluate the effectiveness, usability, and performance of the proposed Retail Management System.
* To prepare and present comprehensive documentation for the developed system and research process.

# Technologies & Resources

This chapter summarizes the tools, platforms, programming languages, frameworks and other resources utilized in developing the project. It explains the reasons for choosing each technology and how they help meet the Project goals and requirements.

## 3.1. Technologies Used

### 3.1.1. Technology Stack Overview

* **Front-End Technologies**

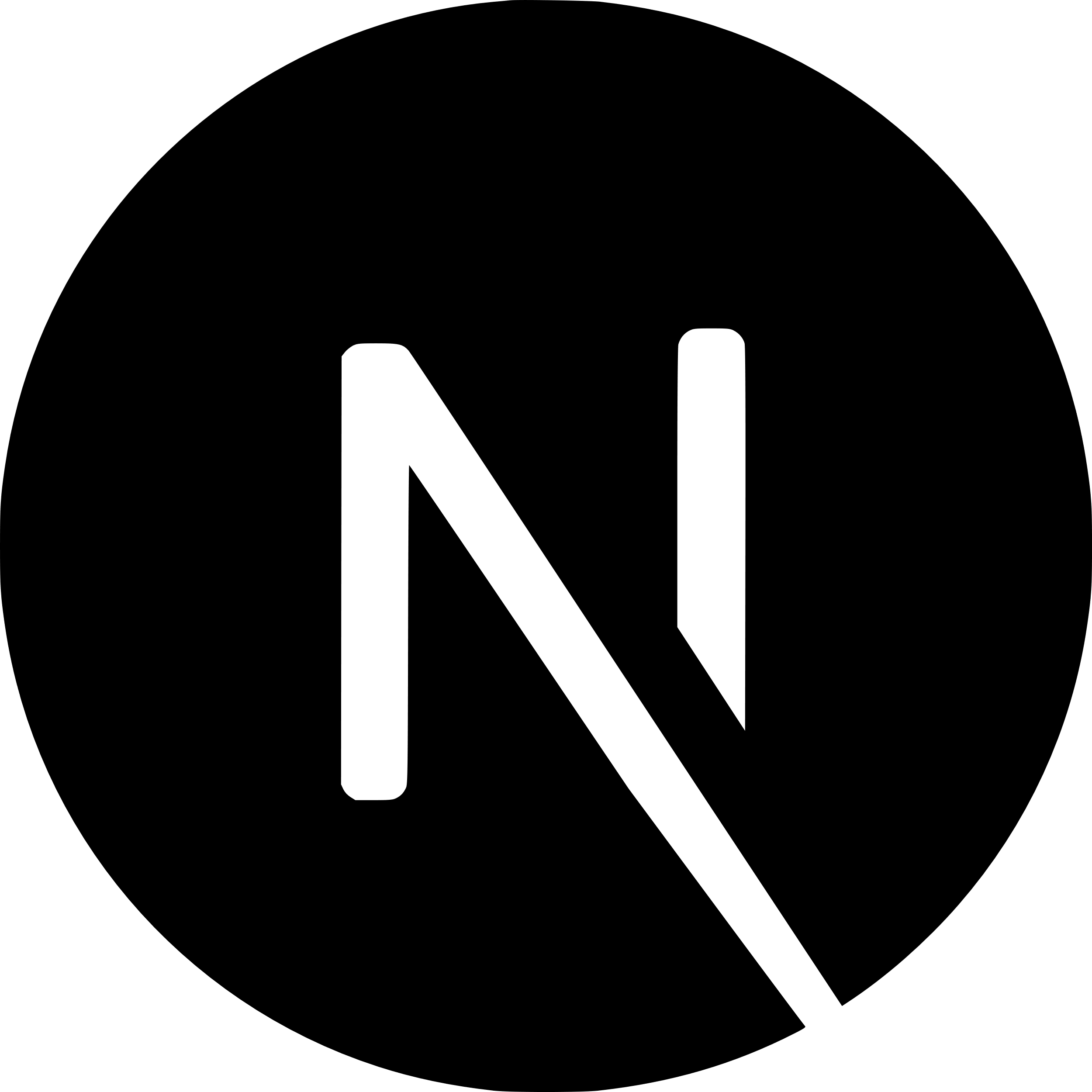
This project has Front-End used Next.js, a React-based framework, for user interface development.

Figure : Next.js

* **Back-End Technologies**

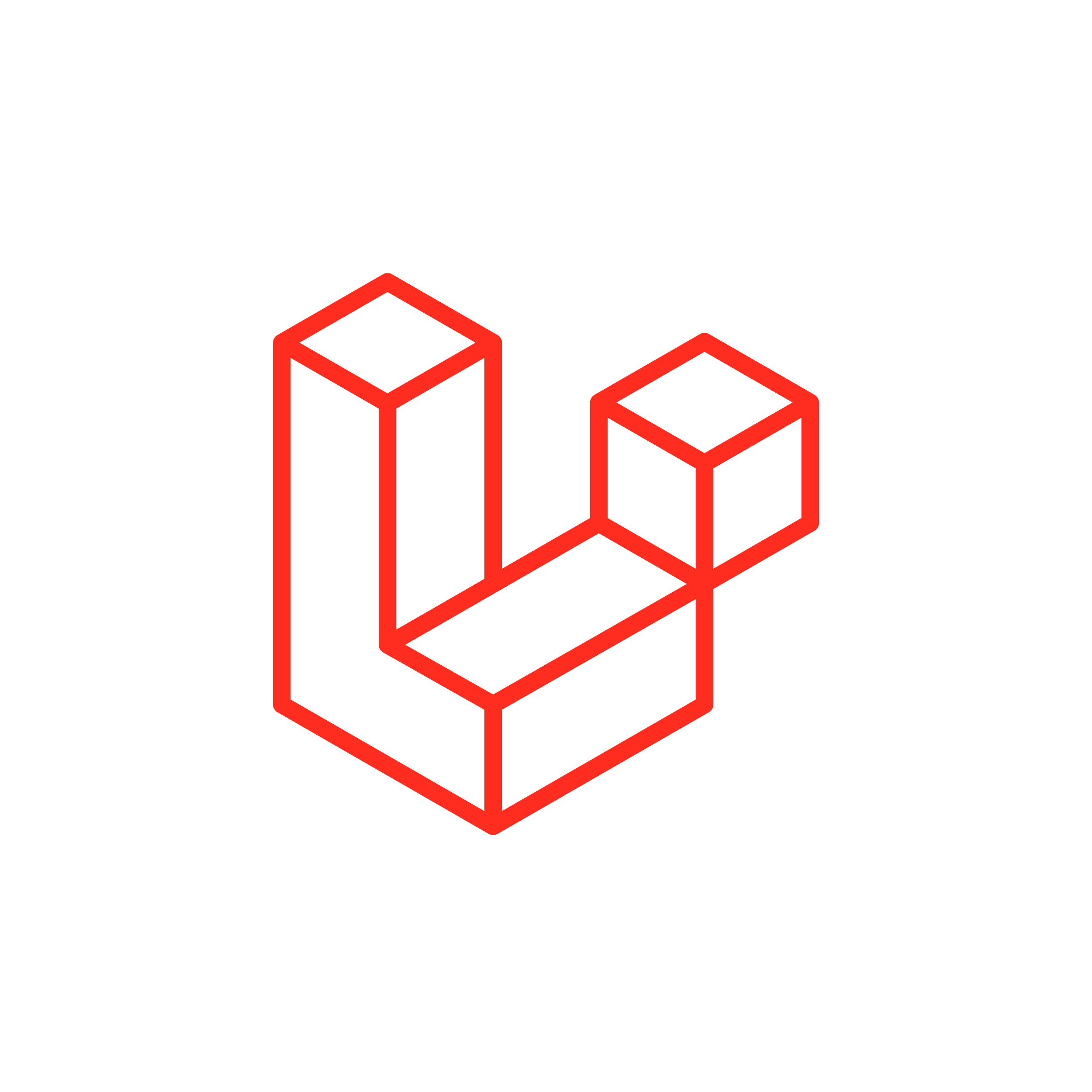
The back-end logic is built using Laravel, a robust PHP framework.

Figure : Laravel

* **Database Management**

A blue and white elephant

AI-generated content may be incorrect.PostgreSQL is the primary database for data storage and retrieval.

Figure : PostgreSQL

* **Additional Tools**

Several third-party libraries, APIs, and tools like Tailwind CSS for styling, shadcn/ui components for UI, and Git for version control are also incorporated.

A logo with blue waves

AI-generated content may be incorrect.A white lines on a black background

AI-generated content may be incorrect.

Figure : Tailwind CSS

Figure : ShadCn

### 3.1.2. Detail Description of Each Technology

* **Next.js (Front-End Framework)**

Next.js is ideal for building the front-end of Ruby Hardware's Retail Management System because it delivers fast, responsive, and SEO-friendly user interfaces. Its combination of server-side rendering (SSR) and client-side rendering (CSR) ensures smooth performance for critical retail functions like real-time product lookup, inventory browsing, and customer interaction. This helps staff work efficiently with minimal loading times, enhancing the user experience.

* **Laravel (Back-End Framework)**

Laravel is used on the back end to handle core business logic, such as processing sales, managing inventory data, handling employee roles, and storing supplier information. Its secure and well-structured architecture includes built-in features like authentication and API support, essential for a system that handles sensitive business data. Laravel's rapid development capabilities also allow for easier maintenance and future scalability.

* **PostgreSQL (Database)**

PostgreSQL is chosen as the database system because it supports complex data structures and ensures data accuracy through its ACID compliance. This is crucial for managing detailed and transactional retail data such as stock levels, product details, invoices, and customer histories. PostgreSQL also offers strong data integrity and scalability, making it suitable for Ruby Hardware's growing operational needs.

* **Tailwind CSS (Styling Framework)**

Tailwind CSS is used to design a clean, user-friendly interface with minimal effort. Its utility-first approach makes it easy to create responsive layouts tailored to the needs of retail staff, such as intuitive navigation for the POS interface or inventory management screens. This improves usability, reduces development time, and ensures consistent design throughout the system.

* **Git (Version Control)**

Git is critical in maintaining code quality and supporting collaboration throughout development. With features like version tracking, branching, and merging, Git allows the project to evolve safely and efficiently. This ensures that updates and enhancements to the Retail Management System can be rolled out without risking system stability, essential for Ruby Hardware's day-to-day operations.

## 3.2. Resource Requirement

**Hardware Requirements**

* **Development Machine**

A laptop with an Intel i5 processor, 8 GB RAM, and SSD storage running Windows 11 is utilized for coding, testing, and deployment tasks. This configuration provides sufficient performance for typical web application development requirements.

* **Cloud Server**

A cloud-based server hosted on Microsoft Azure will be used to deploy backend services and manage the PostgreSQL database. Azure ensures high availability, scalability, and robust performance for production workloads.

**Software Requirements**

* **Operating System**

Development is conducted on Windows 11, providing a stable and compatible environment for software development tools and frameworks.

* **Programming Languages**

JavaScript - Utilized for frontend development using the Next.js framework.

PHP - Employed for backend logic through the Laravel framework.

* **Frameworks & Libraries**

Next.js - Powers the frontend application architecture with support for server-side rendering and routing.

Laravel - Manages backend development, providing robust API handling and business logic implementation.

Tailwind CSS and shadcn/ui - Enhance the user interface with utility-first styling and pre-built UI components.

* **Database**

PostgreSQL - A reliable relational database system used to store and manage structured data.

* **Development Tools**

Visual Studio Code - Serves as the primary IDE for writing, debugging, and organizing code.

Postman - Used for testing and debugging RESTful APIs.

Git - Enables version control for managing source code and tracking changes during the development process.

GitHub - Used as a personal code repository and backup system throughout the project lifecycle.

**Data and Information Resources**

* **User Data**

Simulated user data is generated to test functionalities such as user authentication, role-based access, and transaction logs. This includes anonymized customer records, employee profiles, and sales histories to replicate realistic usage scenarios without compromising the privacy of real data.

* **Inventory & Sales Data**

Sample datasets are created to simulate inventory items, stock levels, pricing, and sales transactions. These synthetic datasets help validate inventory management, billing processes, and real-time stock tracking logic within the system.

* **Learning Materials**

YouTube tutorials were followed to understand the integration of Next.js (Frontend) with Laravel (Backend) and general development practices. These tutorials include:

* How to Connect Laravel with Next.js (OrcDev, 2024)
* Laravel API + Next.js Frontend (Tutor, 2025)
* **Documentation**

API documentation was created to outline the structure of all backend endpoints, including authentication, product CRUD operations, and transaction handling.

Official documentation from Laravel, Next.js, PostgreSQL, and Tailwind CSS was referred to during development for framework-specific best practices.

Guides and manuals for third-party libraries such as shadcn/ui, Jest, and Postman were used to understand component usage, testing strategies, and API debugging workflow.

**Cloud and Hosting Resources**

* **Web Hosting**

Both the frontend (Next.js) and backend (Laravel) applications will be deployed on Microsoft Azure. Azure provides a robust and scalable cloud environment that supports multiple runtimes, ensuring high availability, automatic scaling, and efficient performance for production-grade web applications.

* **Database Hosting**

A PostgreSQL database will be hosted on Azure Database for PostgreSQL, offering enterprise-grade performance, automated backups, and strong security features to manage structured application data reliably.

* **Content Delivery Network (CDN)**

Azure CDN will be used to deliver static content across global edge nodes, enhancing the application's load speed and responsiveness by minimizing latency for end users in different regions.

# 4. Methodology & Work plan

## 4.1. Methodology

This chapter describes the methodology and work plan to be followed to ensure the successful completion of the project. Choosing the right development methodology is crucial for effective project management, timely delivery and the ability to adapt to changes throughout the development process. For this project, the Agile methodology was chosen due to its iterative and flexible nature, which facilitates continuous improvement through regular feedback and incremental feature delivery.

A diagram of software development

AI-generated content may be incorrect.Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is based on the Agile Manifesto, a set of principles that prioritize individuals and interactions, working software, customer collaboration, and responding to change. Agile is an iterative and incremental approach that emphasizes delivering a working product quickly and frequently. It involves close collaboration between the development team and the customer to ensure the product meets their needs and expectations (geeksforgeeks, 2025). The project will be divided into sprints lasting 2 to 3 weeks each, allowing for phased development, testing, and deployment of features.

Figure : Agile (geeksforgeeks, 2025)

## 4.2. Major Phases of the Project

The development of the **Retail Management System for Ruby Hardware** was executed in six structured phases. Each phase had defined objectives, specific tasks, selected technologies, and measurable deliverables to ensure a smooth and successful implementation process.

**Phase 1: Planning and Requirements Gathering**

* **Objective**

The primary objective of this phase was to establish a clear understanding of Ruby Hardware's business operations, define the project scope, and capture all technical and functional requirements necessary for system development.

* **Tasks Performed**

Conducted stakeholder interviews and requirement workshops with Ruby Hardware staff to understand their day-to-day operations, bottlenecks, and software expectations.

Gathered detailed functional requirements (example: sales tracking, inventory management) and non-functional requirements (example: performance, security, scalability).

Identified required data entities and third-party integration points (example. payment gateways or supplier databases) and assessed the implementation and data migration risks.

* **Deliverables**

A comprehensive Project Requirements Specification Document outlining features, data models, and business rules.

A Project Plan including task timelines, milestones, and resource allocation strategy.

* **Technologies Used**

Microsoft Teams - enabled real-time communication and virtual meetings with stakeholders.

Google Docs - allowed collaborative documentation creation and feedback collection.

**Phase 2: System Design**

* **Objective**

This phase aimed to translate the gathered business and technical requirements into a comprehensive system design that ensures scalability, maintainability, performance, and user-friendliness.

* **Tasks Performed**

A relational database schema was designed using PostgreSQL to support core modules such as inventory management, billing, customer records, employee profiles, and transactional logs.

Low-fidelity wireframes created to illustrate key user interfaces, navigation paths, and responsive layouts to enhance device usability.

The backend architecture was defined, including RESTful API endpoints, user authentication flow, middleware logic, and data exchange between client and server layers.

* **Deliverables**

A detailed System Architecture Document was produced, including architecture diagrams and component interactions.

A normalized PostgreSQL database schema was created to ensure data consistency, optimized querying, and referential integrity.

UI Wireframes and Prototypes were developed to visually represent the layout and design of major screens.

An API Specification Document was prepared endpoint URLs, methods (GET, POST, etc.), request/response structures, and authentication requirements.

* **Technologies Used**

PostgreSQL – used to designing and implementing the relational database structure with tables, relationships, and constraints.

Figma – used for designing user interfaces, wireframes, and prototyping the user experience.

GitHub - used for version control, design documentation collaboration and storage of schema definitions and API specifications.

**Phase 3: Development**

* **Objective**

The goal of this phase is to develop and integrate the core components of the system, the frontend, backend, and database - to create a fully functional web application tailored for the retail operations of Ruby Hardware.

* **Tasks Performed**

A responsive user interface is designed and implemented using Next.js, ensuring smooth navigation, real-time updates, and user-friendly experience for retail staff managing products, processing sales, and tracking transactions.

A robust backend is developed using Laravel, focusing on building secure RESTful APIs, managing user authentication and authorization, and embedding key business logic for inventory, billing, and employee functions.

A PostgreSQL database is set up and structured by applying normalization principles to create efficient and consistent data tables for managing stock levels, customer records, sales invoices, and employee details.

All system layers frontend, backend, and database are integrated through well-defined APIs and data validation mechanisms, enabling smooth data exchange and system reliability.

* **Deliverables**

A fully developed and operational retail management system with integrated components supporting real-world business processes.

Successfully tested and deployed API endpoints, ensuring accurate communication between Next.js and the PostgreSQL-powered backend.

All source code was organized, documented, and uploaded to a GitHub repository, enabling efficient version tracking for future updates.

* **Technologies Used**

Next.js builds the front end, combining server-side rendering and client-side interactivity for a seamless user experience.

Laravel is used to develop backend APIs, enforce security protocols, and handle business logic across system modules.

PostgreSQL is selected as the relational database for its reliability, scalability, and support for structured retail data.

Visual Studio Code for the main development environment for coding, debugging, and managing files throughout the project.

GitHub is used for version control, team collaboration, and tracking changes across all development phases.

**Phase 4: Testing and Quality Assurance**

* **Objective**

This phase will focus on thoroughly testing the system to verify that it meets all functional requirements and performance standards while identifying and addressing any defects before the final deployment.

* **Tasks Performed**

Unit tests will be conducted to ensure each component worked correctly, followed by integration tests to check how different modules interacted and comprehensive system tests to validate the overall application behavior.

User Acceptance Testing (UAT) will be performed with selected Ruby Hardware staff to confirm the system's usability and effectiveness in real-world scenarios.

Issues related to system performance, UI inconsistencies, and potential security risks will be detected and fixed to enhance the application's stability and reliability.

* **Deliverables**

Comprehensive test reports will document test coverage, defect tracking, and resolutions.

A thoroughly tested, deployment-ready application will receive approval from end-users.

* **Technologies Used**

Postman is responsible for API testing to verify backend endpoints and data flow.

Jest will be used to automate tests of frontend features and user interface behavior.

Trello will be manage bug tracking, test case documentation, and issue resolution workflows.

**Phase 5: Deployment**

* **Objective**

This phase will focus on deploying the completed system into a secure, scalable production environment, ensuring its stability and accessibility for all users.

* **Tasks Performed**

The application will be hosted on cloud services like Azure, guaranteeing high availability and easy scalability as user demand grows.

Automated backups, disaster recovery plans, and SSL certificates will be configured to ensure data security and protect user transactions.

Monitoring and analytics tools will be set up to track system performance and user activity and detect issues in real time.

* **Deliverables**

A live, fully functional system will be accessible to Ruby Hardware employees.

A secure infrastructure with defined user access roles and continuous performance and security monitoring will be established.

* **Technologies Used**

Azure will be used to host web applications and backend services.

PostgreSQL as the production database, ensuring reliable and consistent data storage and management.

Google Analytics will capture user engagement and system usage data for ongoing improvements.

**Phase 6: Maintenance and Iteration**

* **Objective**

This phase will focus on providing continuous support for the system, keeping it running smoothly, gathering user feedback, and implementing updates to meet changing business requirements.

* **Tasks Performed**

Server health, user activity, and system logs will be regularly monitored to identify and resolve any issues quickly before they impact users.

User feedback will be gathered through surveys and support requests to understand needs and improve the system accordingly.

Regular updates will be rolled out, including new features, bug fixes, and security enhancements based on audit results and user input.

* **Deliverables**

Updated versions of the application will be released periodically with improvements and resolved issues.

Reports on system performance and user feedback will help shape the future development roadmap.

* **Technologies Used**

GitHub to manage code versions and track all changes efficiently.

New Relic will be used to monitor application performance and diagnose issues in real time.

Various survey tools will be used to systematically collect and analyze user feedback.

## 4.3. Feasibility Study

A Feasibility Study in Software Engineering is an evaluation process that helps determine whether a proposed project or system is practical and achievable. It is one of the key stages in the Software Project Management lifecycle. The feasibility study evaluates the practical benefits of developing the software for the organization. This analysis examines various factors, including whether the software can be successfully developed and implemented, as well as the value the project will bring to the organization. The goal is to ensure that the project is worthwhile before committing significant time and resources to it. (geeks for geeks, 2024)

* **Technical Feasibility**

The system's technical foundation is well-suited to the Project goals and requirements. It utilizes modern, efficient technologies such as Next.js for the frontend, Laravel for the backend, and PostgreSQL for the database, all of which are compatible and widely supported. Development and testing are conducted on a laptop with an Intel i5 processor, 8GB RAM, and SSD storage, which is sufficient for typical web application development. Hosting and deployment are managed using Microsoft Azure, which provides high availability, scalability, and seamless integration with the selected technology stack. Additionally, Visual Studio Code and GitHub offer robust environments for code management and collaboration, ensuring the technical feasibility of the project.

* **Economic Feasibility**

The project is economically viable as it primarily relies on free and open-source tools, reducing overall development costs. Technologies such as Laravel, PostgreSQL, Visual Studio Code, Git, and GitHub do not require licensing fees, making them ideal for student or small business projects. Furthermore, hosting costs are kept to a minimum by utilizing Azure's student or free-tier cloud services. Optional tools, such as Google Forms for feedback collection, are also free, making the entire project cost-effective and manageable within available resources.

* **Operational Feasibility**

Operationally, the system is designed to address the specific needs of Ruby Hardware, including inventory tracking, sales management, and reporting. The user interface is intuitive and user-friendly, ensuring that end-users, such as cashiers and administrators, can use the system effectively with minimal training. Role-based access control (RBAC) is implemented to maintain operational security and data integrity. Additionally, structured feedback collection through tools like Google Forms enables iterative improvement based on real user input, reinforcing the system’s practicality in a live retail environment.

## 4.5. Major Milestones & Deliverables

The project is organized into five major milestones, each representing a crucial stage in the development of the Retail Management System for Ruby Hardware.

* **Milestone 1 – Complete Project Planning**

This milestone involved preparing the project requirements document and a detailed work plan, which defined the project scope, objectives, and resources. This phase set a clear direction for the project and ensured all requirements were documented before starting design and development.

* **Milestone 2 - Complete System Design**

This stage included creating the system architecture, user interface wireframes, and API specifications, providing a transparent design framework for development. The finalized design ensured all functional and technical requirements were clearly outlined and approved.

* **Milestone 3 - Complete Development**

By this milestone, the entire application is built, integrating the Next.js frontend, Laravel backend, and PostgreSQL database, with the source code properly documented and version controlled. This phase focuses on implementing all planned features according to the approved design and ensuring the quality of the code.

* **Milestone 4 - Complete Testing**

This phase consists of thorough testing, including unit, integration, system, and User Acceptance Testing (UAT), to ensure the application meets all requirements and is free from critical bugs. Testing verified functionality, performance, and reliability to confirm readiness for deployment.

* **Milestone 5 - Complete Upcoming**

The final milestone marks the live launch of the system on a secure cloud platform with backup and monitoring systems configured to maintain performance and security. This ensures the system is accessible to end users and supported by proper operational procedures.

**Milestone Table**

Table : Milestones Timeline Table

|  |  |  |
| --- | --- | --- |
| **Milestones** | **Description** | **Estimated Timeframe** |
| 1st Milestone - Project Planning | Define scope, objectives, and resources | 4 Weeks |
| 2nd Milestone - Design | Develop UI wireframes, system architecture, and APIs | 6 Weeks |
| 3rd Milestone -Development | Build frontend, backend, and database integration | 14 Weeks |
| 4th Milestone - Testing | Perform unit, integration, system, and UAT testing | 3 Weeks |
| 5th Milestone - Deployment | Deploy system to cloud, configure backup and monitoring | 4 Weeks |

## 4.5. Contingency Plan

A diagram of a contingency planning

AI-generated content may be incorrect.A contingency plan is a course of action designed to help an organization respond effectively to a significant future incident, event, or situation that may or may not happen. Often referred to as a "Plan B" or backup plan, it provides an alternative course of action if expected results fail to materialize. Contingency planning is a key component of business continuity, disaster recovery, and risk management. (Kirvan, 2022)

Figure : Contingency Planning

During this project, several potential risks and their management strategies are identified to maintain consistency. One significant risk was the possibility of development delays caused by unexpected challenges or workload fluctuations. To address this, the contingency plan includes extending the time allocated for specific tasks or adjusting the project schedule, ensuring that quality is not compromised despite slower progress. As this was an individual project, adding more developers is not an option, so effective time management and prioritization are essential.

Another risk concerns the integration of front-end and back-end components, which could result in functional inconsistencies or bugs. To mitigate this, additional time is explicitly set aside for integration testing to verify communication between the user interface and server logic thoroughly. Close attention is paid to debugging and iterative testing to resolve issues promptly.

Finally, data security and privacy breaches are identified as critical risks, mainly due to the sensitive nature of retail and customer information. The plan included implementing robust security measures such as strong authentication, data encryption, and regular security audits to detect vulnerabilities early. These precautions aim to protect user information and maintain trust by proactively safeguarding data and applying best security practices.

## 4.6. Gantt Chart

Figure : Gantt Chart

# Proposed Solution

This chapter outlines the guiding principles, data protection strategies, and safety mechanisms incorporated into the **Retail Management System for Ruby Hardware**. The solution is specifically designed to meet the operational demands of a hardware retail environment while prioritizing security, ethical data handling, and legal compliance. The system aims to deliver a reliable, secure, and user-friendly platform for managing retail operations efficiently by addressing these foundational aspects.

## 5.1 Ethics

Ethical practices are fundamental to the structure and operation of the system. The system is developed to ensure responsible data handling, maintain transparency, and uphold user trust. Core ethical measures implemented include

* **Clear Privacy Policies**

A concise and accessible privacy statement explains how data related to customers, sales, and operations is collected, processed, and stored within the application. (European Commission, 2018)

* **Data Access and Control**

Administrative-level users have full access to view, update, and delete relevant records, including customer details and transaction history. This ensures accountability and control over business-critical data.

* **Restricted Data Sharing**

No customer or business information is shared with external entities without explicit consent, ensuring complete confidentiality.

**The following principles guide the ethical framework**

* **Privacy Protection**

All personal and business-related data is treated with strict confidentiality. (Floridi, 2016)

* **Data Accuracy**

Inventory, transaction, and customer data are maintained with high integrity and consistency.

* **Transparency**

Information on data processing, storage, and protection measures is available to ensure transparent user experience.

## 5.2 Data Protection

Robust data protection strategies are essential for securing sensitive retail data, including customer profiles, inventory records, and financial transactions. To mitigate risk and ensure compliance, the system integrates the following data security measures:

* **Encryption**

All sensitive data is encrypted both in transit-using HTTPS with TLS and at rest within the PostgreSQL database. This ensures that data remains secure during transfer and storage. (OWASP, 2023)

* **Role-Based Access Control (RBAC)**

Permissions and access levels are clearly defined, allowing specific actions only to authorized users (e.g., cashier, administrator). This minimizes exposure to unauthorized access or manipulation. (OWASP, 2023)

* **Comprehensive Audit Logs**

All system events, such as logins, inventory updates, and transaction entries, are logged to enable activity tracking and support forensic investigations in case of anomalies. (OWASP, 2023)

**Compliance with international and local data protection laws is also taken into consideration**

* **GDPR Compliance**

The system aligns with General Data Protection Regulation standards to accommodate potential expansion to European Union markets. (European Commission, 2018)

* **Local Regulations**

Data handling practices follow applicable laws and privacy regulations within the region (e.g., Sri Lanka), ensuring legal compliance. (European Commission, 2018)

## 5.3 Safety

To safeguard the system against cyber threats, data breaches, and other operational risks, a range of security mechanisms are built into the software

* **Secure Authentication**

User accounts are protected through strong password policies and optional multi-factor authentication (MFA) for elevated access levels, such as administrative accounts. (Grassi, et al., 2017)

* **Encrypted Storage**

Sensitive records, including supplier details, customer contact information, and financial logs, are encrypted to prevent unauthorized access or exposure. (OWASP, 2023)

* **Periodic Security Reviews**

The codebase is subject to regular audits, including manual testing and vulnerability assessments, to proactively identify and remediate security flaws. (OWASP, 2023)

* **Secure Development Practices**

Secure coding standards are enforced throughout development. Input fields are validated to prevent injection attacks, SQL queries use prepared statements, and third-party packages are managed responsibly. (OWASP, 2023)

## 5.4 Suggested Starting Point

A structured approach was adopted to initiate development efficiently and ensure alignment with business objectives. Key steps include

* **Requirements Analysis**

Operational needs of the retail environment are examined in detail, focusing on core functions such as inventory control, sales processing, customer records, and report generation.

* **System Requirements Definition**
* **Functional Requirements**

Core functionalities include user authentication, product categorization, invoice creation, stock-level alerts, and comprehensive reporting tools.

* **Non-Functional Requirements**

Performance, responsiveness, data security, and usability are prioritized to enhance the overall system experience.

* **Project Planning and Milestone Mapping**

A detailed roadmap was created, outlining development phases such as planning, interface design, backend integration, testing, and final deployment. Each phase includes clear deliverables and timelines.

* **Technology Stack Selection**
* **Frontend**

The system is developed using Next.js, styled with Tailwind CSS and shadcn/ui for a modern and responsive interface.

* **Backend**

The backend is powered by Laravel, which was selected for its robust features and scalability.

* **Database**

PostgreSQL is used for its reliability and security in managing complex datasets.

* **Development Tools**

GitHub is utilized for version control and collaborative code management.

**Prototype Development**

Early-stage wireframes and a working prototype featuring the POS module and inventory dashboard are developed to validate key functions and gather stakeholder feedback before full-scale implementation.

# Discussion*.*

Developing a software solution such as the **Retail Management System for Ruby Hardware** requires careful consideration of legal, ethical, societal, and security concerns. Addressing these factors supports compliance with regulatory standards, strengthens system reliability, promotes fairness, builds stakeholder trust, and reduces potential risks. Integrating these considerations into the development life cycle ensures the system is robust, secure, and socially responsible.

## 6.1. Legal Issues

* **Data Protection Laws**

Compliance with regulations such as the **General Data Protection Regulation (GDPR)** (European Commission, 2018) and **the California Consumer Privacy Act (CCPA)** (California Legislative Information, 2018) is essential. These laws mandate responsible handling of customer data collected through sales transactions, loyalty programs, or user accounts. The system's architecture incorporates legal requirements such as obtaining explicit consent, defining data retention periods, and implementing secure data disposal procedures.

**Example in the System**

Customer information and transaction history are collected with explicit user consent, and records can be securely deleted upon request or after a defined period.

* **Intellectual Property (IP) Rights**

Retail software often uses original UI/UX designs, proprietary business logic, and possibly user-generated content like product reviews. To prevent IP conflicts, clear ownership of digital assets-both internal (software code, design) and external (business data)-is defined.

**Example in the System**

All modules, interfaces, and features are custom developed for Ruby Hardware and documented to establish ownership. Any third-party libraries are used in accordance with their licenses.

* **Accessibility Compliance**

The system considers legal standards such as **WCAG 2.1** (W3C (World Wide Web Consortium), 2018) to ensure it is usable by individuals with disabilities. This includes support for screen readers, high-contrast themes, keyboard navigation, and semantic HTML structure.

**Example in the System**

Accessible UI components ensure the POS system is operable by visually or physically impaired staff without relying on mouse input alone.

## 6.2. Ethical Issues

* **User Privacy Consent**

Ethical design practices require that users are fully informed about how their data is collected and used. Retail systems often handle sensitive personal data like contact information, purchase history, and payment details. (Floridi, 2016)

**Example in the System**

A clear privacy notice is displayed within the dashboard interface, and user consent is required during account creation or when subscribing to digital receipts or newsletters.

* **Bias and Fairness in Algorithms**

Although this system does not rely heavily on advanced AI, rule-based recommendations (e.g., low-stock alerts and popular product filters) are implemented without bias. Fairness is maintained in sales reporting and employee performance tracking to avoid misinterpretation or discrimination. (Mittelstadt, et al., 2016)

* **Ethical Use of Technology**

Ethical development practices guide the handling of analytics and automation features. Business intelligence tools support informed decision-making without violating customer rights or manipulating user behavior for profit. (Floridi, 2016)

**Example in the System**

Sales trends and customer preferences are aggregated without linking behavior directly to individual users unless explicitly permitted.

## 6.3. Societal Issues

* **Accessibility and Inclusivity**

A system tailored for local hardware businesses like Ruby Hardware must accommodate users with varying levels of digital literacy. A simplified and intuitive interface ensures that staff members with a minimal technical background can perform daily operations effectively. (Jaeger, 2012)

**Example in the System**

Icons, color-coded alerts, and tooltips enhance usability. A localized language option can also be introduced for regional operators.

* **Knowledge Equity and Educational Impact**

Retail staff and business owners benefit from training resources and user manuals integrated into the system. This promotes digital literacy and helps bridge the knowledge gap, especially in small or rural businesses. (Selwyn, 2016)

* **Privacy Concerns and Trust**

Transparent communication about the purpose of each data field and how information is protected fosters trust. Ensuring that users are never misled or monitored without consent is key to social responsibility. (Nissenbaum, 2024)

**Example in the System**

The system explicitly states why customer details are collected at checkout (e.g., for receipt, delivery, or loyalty) and includes options to opt out.

## 6.4. Security Issues

* **User Authentication and Account Security**

Role-based authentication is implemented to distinguish between administrators, cashiers, and inventory managers. Multi-factor authentication (MFA) adds an extra layer of protection for sensitive access. (Grassi, et al., 2017)

**Example in the System**

Admin accounts require password and OTP verification before granting access to financial reports or staff management tools.

* **Data Protection and Encryption**

Sensitive data, such as supplier contacts, product costs, and transaction logs, is encrypted during transmission (via HTTPS/TLS) and securely stored in the PostgreSQL database using hashing and encryption standards.

* **Regular Security Audits and Vulnerability Assessments**

Routine code reviews, penetration testing, and security audits help maintain software integrity. Dependencies are regularly checked for vulnerabilities using **Laravel Security Checker**.

**Example in the System**

Security logs track login attempts, changes to pricing, and unauthorized data access attempts, enabling early detection and mitigation of threats.

By addressing these legal, ethical, societal, and security issues early in development, the **Retail Management System for Ruby Hardware** is designed to operate responsibly and securely, delivering long-term value to the business and its customers. The approach aligns with modern development standards and emphasizes the importance of trust, compliance, and inclusiveness in digital retail environments.

# References

California Legislative Information, 2018. *California Legislative Information.* [Online]   
Available at: https://leginfo.legislature.ca.gov/faces/codes\_displayText.xhtml?lawCode=CIV&division=3.&title=1.81.5.&part=4.&chapter=&article=  
[Accessed 27 05 2025].

European Commission, 2018. *Europien Commision.* [Online]   
Available at: https://commission.europa.eu/law/law-topic/data-protection\_en  
[Accessed 24 05 2025].

Floridi, L., 2016. *The Ethics of Information.* s.l.:Oxford University Press.

Gartner, 2023. *Market guide for retail management software.* [Online]   
Available at: https://www.gartner.com/  
[Accessed 20 05 2025].

geeks for geeks, 2024. *geeks for geeks.* [Online]   
Available at: https://www.geeksforgeeks.org/types-of-feasibility-study-in-software-project-development/  
[Accessed 28 05 2025].

geeksforgeeks, 2025. *geeksforgeeks.* [Online]   
Available at: https://www.geeksforgeeks.org/software-engineering-agile-software-development/  
[Accessed 24 05 2025].

Goyal, A. & Gupta, . V., 2020. : Automation in retail: Improving productivity and customer experience. *International Journal of Business Research.*

Grassi, P., Garcia, M. & Fenton, J., 2017. *NIST Special Publication 800-63-3: Digital Identity Guidelines,* s.l.: U.S. Department of Commerce, National Institute of Standards and Technolog.

Jaeger, P., 2012. *Disability and the Internet: Confronting a Digital Divide.* s.l.:Lynne Rienner Publishers.

Kirvan, P., 2022. *techtarget.* [Online]   
Available at: https://www.techtarget.com/whatis/definition/contingency-plan  
[Accessed 26 05 2025].

Mittelstadt, B., Allo, P., Taddeo, M. & Wachter, 2016. The ethics of algorithms: Mapping the debate. *Big Data & Society,* 3(2), pp. 1-21.

Nissenbaum, H., 2024. Privacy as Contextual Integrity. *Washington Law Review,* 79(1), pp. 119-157.

OrcDev, 2024. *Laravel Backend + Next.js Frontend: The How-To Guide,* s.l.: OrcDev.

OWASP, 2023. *OWASP.* [Online]   
Available at: https://owasp.org/www-project-top-ten/  
[Accessed 24 05 2025].

Raj, A. & Singh, R., 2021. Challenges in retail automation for SMEs. *Journal of Retail and Consumer Services.*

Selwyn, N., 2016. *Education and Technology: Key Issues and Debates.* s.l.:Bloomsbury Academic.

Sommerville, I., 2016. *Software Engineering.* 10th ed. ed. s.l.:Pearson Education.

Tutor, O. W., 2025. *Build a Full-Stack CRUD App with Next.js & Laravel 12 | Full 3-Hour Course,* s.l.: s.n.

W3C (World Wide Web Consortium), 2018. *W3C (World Wide Web Consortium).* [Online]   
Available at: https://www.w3.org/TR/WCAG21/  
[Accessed 27 05 2025].

# Appendices

**Appendix A – System Key Functional Modules**

Table : Appendix A - System Key Functional Modules

|  |  |
| --- | --- |
| **Module** | **Description** |
| Dashboard | Provides an overview of total clients, earnings, product count, recent activities, and visual sales/payment charts for real-time insights. |
| Clients | |  | | --- | |  |  |  | | --- | | Manages customer profiles, contact details, transaction history, and credit limits for effective customer relationship management. | |
| Sales | Supports creation and management of invoices, quotations, sales orders, and returns. Integrated with inventory and client modules for seamless sales operations. |
| Employees | Maintains employee data including roles, attendance, performance, and payroll. Also manages user access and permissions. |
| Suppliers | Handles supplier information, purchase history, payment terms, and communication details to streamline vendor management. |
| Purchases | Facilitates the creation and tracking of purchase orders, goods receipts, and supplier invoices to ensure timely stock replenishment. |
| Reports | Generates customizable reports across modules, with export options (PDF, Excel) for monitoring and compliance. |
| Inventory | Manages product stock, categories, quantities, and movement to ensure accurate inventory control and real-time updates. |
| Settings | Allows configuration of user roles, system preferences, permissions, tax rates, and other key parameters to tailor the system. |
| POS (Point of Sale | Offers a real-time sales interface with features like barcode scanning, billing, discounts, and support for multiple payment methods. |
| AI (Chatbot) | Assist customers by retrieving and analyzing reports, identifying trends, and proposing data-driven business solutions to improve performance and decision-making. |

**Appendix B – Database Schema Design**

Table : Appendix B - Database Schema Design

|  |  |
| --- | --- |
| **Table** | **Description** |
| Client Table | Stores client details such as full name, NIC, contact info, photo, address, client type, invoice method, and status |
| Product Table | Stores product details such as name, category, brand, pricing, tax, discounts, margin, supplier reference, and inventory tracking. |
| Sales Table | Store sales transactions including client ID, invoice number, date, and associated product details. |
| Employee Table | Store employee data such as name, NIC, DOB, contact info, photo, role, and system access permissions. |
| Purchase Table | Store purchase details such as supplier ID, purchase number, date, and product references |
| Supplier Table | Stores supplier details such as business name, contact details, NIC, address, status, and other relevant identification info. |

**Appendix C – System Users Roles**

Table : Appendix C – System Users Roles

|  |  |
| --- | --- |
| **Role** | **Description** |
| Admin (Hardware Owner / Manager) | Full system access to all modules and features, including data management and access control. |
| Employees | |  | | --- | |  |   Limited access based on permissions. Can perform basic tasks (example:- create invoices, add clients) but edit, or delete data unless authorized. |

|  |
| --- |
|  |