

South East Technological University

Department of Computing & Mathematics

A High-Performance Edge-Native Point of Sale Terminal with Zero Trust Security

FINAL PROJECT REPORT

NAME: Vadym Melnychenko

STUDENT ID: W20108893

SUPERVISOR: Michael McMahon

MODULE: ICT Placement Skills

COURSE: Higher Diploma in Science in Computer Science

April 2026

Declaration of Authenticity

I declare that the work which follows is my own, and that any quotations from any sources (e.g. books, journals, the internet) are clearly identified as such by the use of 'single quotation marks', for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student..... Date

Word Count

This project contains 7833 words.

Statement of Copyright and Intellectual Property

1. Ownership and Copyright

The author and South East Technological University (formerly Waterford Institute of Technology) retain copyright of this project for academic purposes. However, the author remains the sole legal owner of the original concepts, software architecture, and source code associated with the **Wynsum™ RMS** ecosystem and its components, including the Wynsum POS Terminal implementation, except where explicitly referenced otherwise.

2. Proprietary Information and Trade Secrets

This document and the associated software contain proprietary information, trade secrets, and confidential algorithms that are the intellectual property of the author. The submission of this project for academic assessment does not constitute a waiver of any intellectual property rights or a license for commercial exploitation by any third party or the institution.

3. Restrictions on Use All rights reserved.

The use, reproduction, or transmission of any part of this document or the associated source code—whether electronic, mechanical, photocopying, recording, or otherwise—without the prior written consent of the author is strictly prohibited.

4. Limited Academic License

The South East Technological University is granted a non-exclusive, non-transferable license to use this material solely for the purpose of academic evaluation, moderation, and archival record-keeping. Any further distribution or application of the software components (specifically the Wynsum RMS POS Terminal logic) requires a separate licensing agreement with the author.

Abstract

This project delivered the Wynsum POS Terminal, a high-performance Progressive Web App (PWA) engineered as the primary operational interface within an Edge-Native Retail Management System (RMS). The platform was developed to resolve the "operational disconnect" where sophisticated modern marketing requirements, such as recursive discounts and dynamic pricing, often outpace the technical capabilities of legacy infrastructure.

Notably, the system redefines "high performance" within the constraints of strict data consistency required for retail. Unlike standard web applications that rely on client-side caching for speed, **Wynsum POS deliberately eschews traditional caching layers** to prevent data discrepancies and inventory drift. Instead, sub-millisecond responsiveness is achieved through a specialised WebSocket-based JSON-RPC protocol and a high-performance Universal Infinite List with row virtualization, maintaining a consistent 60 FPS while managing datasets exceeding 10,000 items (Forward, 2015).

Adhering to a "Zero Trust" security model, the system migrated all critical financial logic exclusively to the secure server-side Wynsum Hub, effectively establishing a reliable "single source of truth". The final implementation ensures 100% operational functionality with an ultra-low local API response time of <5ms, providing the speed of a local system with the intelligence of a modern cloud platform.

Keywords

Edge-Native Architecture, Zero Trust Security, Retail Management System (RMS), Python/Django, React Progressive Web App (PWA), WebSockets, JSON-RPC, ECDSA Cryptography, IndexedDB, Asynchronous Order Pipeline, Distributed Systems.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Michael McMahon, for his professional mentorship and invaluable guidance in organizing the project workflow. His regular consultations and constructive feedback regarding the planning of development stages were instrumental in helping me maintain a consistent pace across all seven sprints. I particularly value Michael's management expertise and his emphasis on mitigating risks through the effective allocation of tasks, both of which significantly contributed to the successful completion of the Wynsum RMS project.

Table of Contents

Declaration of Authenticity.....	2
Statement of Copyright and Intellectual Property.....	2
Abstract.....	3
Keywords.....	3
Acknowledgements.....	3
Table of Contents.....	4
List of figures.....	5
Glossary.....	5
Introduction.....	7
1.1 Background The Challenge of Modern Retail Operations.....	7
1.2 Project Goal.....	8
1.3 Scope and Technical Architecture.....	9
1.3.1 Backend and Hub Logic (The "Single Source of Truth").....	9
1.3.2 Frontend (The Wynsum POS User Experience).....	10
1.3.3 Communication and "Zero Trust" Terminal Security.....	10
1.4 Key Deliverables: The Wynsum POS Ecosystem.....	11
1.5 Anticipated Technical Challenges.....	13
Research and Analysis.....	15
2.2 Technical Research.....	16
2.3 Market Analysis: Wynsum RMS's Unique Value Proposition.....	18
3. Modelling and Design.....	20
Project Management.....	22
4.1 Methodology: Agile/Scrum Framework.....	23
4.2 Sprint Plan and Phasing.....	23
The Scrum 3 example:.....	25
4.3 Project Completion and Commercial Evolution.....	27
Current Status and Implementation Outcomes.....	28
5.1 Implementation Status.....	28
5.2 Problems Encountered and Resolutions.....	29
5.3 Future Work: Transitioning to Production-Grade Infrastructure.....	30
5.4 What I Learned: Architectural DevOps Upskilling.....	31
AI Disclosure and Methodology.....	33
1. Strategic Research and Technical Benchmarking.....	33
2. Structural Planning and Alignment.....	33
3. Linguistic and Grammatical Refinement.....	33
4. Software Development and Implementation.....	33
Appendices and References.....	35

List of figures

Figure 1 - Entity Relationship Diagram of Wynsum Hub

Figure 2 - Edge-Native System Architecture Overview

Figure 3 - Trello board example

Figure 4 - Custom task tracking list using MS Excel

Glossary

Asynchronous Checkout Pipeline A non-blocking, multi-step transaction processing mechanism implemented in the Wynsum Hub. It allows the system to handle inventory reservation, payment processing, and fiscalization sequentially without blocking the main execution thread of the POS terminal.

CIT (Client Installation Token) A unique identifier used for versioning integrity and authorized deployment. It ensures that the Progressive Web App (PWA) and the Wynsum Hub remain compatible and that only authorized users can install the proprietary interface.

Connectivity Resilience The ability of the POS terminal to remain 100% functional during network instability or total internet failure by utilizing local persistence layers for data caching and synchronization.

Delta-Synchronization An optimized data transfer logic (`handle_sync_items`) that fetches only the changes made since the terminal's last known state. This reduces payload size and ensures data consistency between the local terminal and the central Hub.

ECDSA (Elliptic Curve Digital Signature Algorithm) A cryptographically secure algorithm used for device verification in the project's **Zero Trust** model. It requires the physical POS hardware to sign server-issued challenges using non-exportable hardware keys.

Edge-Native Architecture A specialized infrastructure where core business logic and API services are hosted on a local industrial-grade server (Wynsum Hub) within the store's network. This provides ultra-low response times (<5ms) while maintaining the intelligence of a cloud platform.

IndexedDB A low-level API for client-side storage of significant amounts of structured data. It serves as the primary local database for the Wynsum POS terminal to ensure operational continuity during offline sessions.

JSON-RPC (Remote Procedure Call) A lightweight communication protocol encoded in JSON. A custom JSON-RPC router was implemented in this project to structure request-response consensus over persistent WebSocket streams.

RMS (Retail Management System) A comprehensive software ecosystem (including the Hub and POS terminals) designed to manage retail operations, financial compliance, and revenue optimization.

Single Source of Truth (SSoT) An architectural design principle where all critical financial logic, tax calculations, and transaction records are centralized in the secure server-side Wynsum Hub. This ensures 100% data integrity and eliminates discrepancies between client nodes.

Introduction

1.1 Background The Challenge of Modern Retail Operations

In the contemporary, highly competitive retail and hospitality landscape, marketing strategies evolve rapidly to encompass intricate loyalty programmes, dynamic pricing models, and multi-layered promotional rules. However, a significant operational disconnect exists between these sophisticated marketing requirements and the technical capabilities of legacy infrastructure. Many existing systems underperform, effectively "leaving money on the table" because they cannot measure the true impact of pricing strategies or provide near real-time insights.

A modern Retail Management System (RMS) is now required to serve concurrently as a "system-of-record" and an intelligent "productivity workspace" that enables critical processes amid a fast-changing environment. Despite this, the industry standard often relies on client-side logic for financial calculations, which introduces a high risk of errors—particularly regarding rounding precision, conflicting discount rules, and general data discrepancies between the terminal and the backend. Research indicates that legacy systems requiring constant manual reprogramming are inefficient and limit the growth potential of an organisation.

Furthermore, the digital evolution of the sector demands the integration of disconnected systems into a unified ecosystem to ensure data integrity and enhance the guest experience. This includes the necessity for robust device authentication to protect sensitive financial data, ensuring that only verified, physical hardware can execute critical business logic.

1.2 Project Goal

The core objective of this project was the development of the Wynsum POS Terminal, a high-performance, edge-connected interface within the broader Wynsum Retail Management System (RMS). Following the industry definition of a "modern RMS," the terminal was engineered to serve concurrently as an authoritative "system-of-record" for financial compliance and an intelligent, high-speed "productivity workspace" that enables critical retail workflows in a fast-changing environment.

To achieve these goals, the implementation of the POS terminal focused on three primary strategic pillars:

1. Ensuring Absolute Data Integrity through a "Zero Trust" Terminal Architecture:

Recognising that cybersecurity is a non-negotiable prerequisite for modern financial environments, the project implemented a "Zero Trust" security model specifically regarding the frontend interface. The system was fundamentally designed to treat the client-side POS terminal as a potentially compromised environment where critical business logic should never be executed. Consequently, all heavy financial calculations—including sequential or recursive discounts, dynamic product bundles, and multi-layered tax calculations—were migrated exclusively to the secure server-side Wynsum Hub. To verify this trust, the terminal utilizes a cryptographic challenge-response protocol, signing server-issued challenges with non-exportable hardware keys to ensure that only registered, physical hardware can interface with the backend. This architecture successfully eliminates the mathematical rounding errors and price manipulation risks historically associated with traditional client-side POS systems.

2. Bridging the Gap between Marketing Complexity and Terminal Operational Speed:

The implementation prioritised resolving the common "operational disconnect" where complex marketing requirements often outpace the technical capabilities of legacy POS hardware. The terminal functions as a high-performance Progressive Web App (PWA), acting as a "thin client" that remains responsive even under high-concurrency loads. By leveraging an Asynchronous Checkout Pipeline (integrating inventory, payment, and fiscalisation steps), the terminal can process complex transactions without blocking the main execution thread. (Forward, 2015)

Furthermore, to handle the massive datasets typical of modern retail, the interface employs a Universal Infinite List with row virtualization, allowing operators to browse over 10,000 items with a consistent performance of 60 FPS. This ensures that sophisticated business logic does not create bottlenecks at the point of service, maintaining a low-latency response time of <5ms via a persistent WebSocket connection.

3. Maximising Revenue Optimization through a Unified Terminal Workspace:

Beyond simple transaction processing, the Wynsum POS was engineered to be an essential tool for revenue optimization and capturing "total guest value". Unlike standard POS systems that offer limited visibility, the Wynsum terminal serves as the primary entry point for near real-time insights.

Through its integration with the unified backend dashboard, it enables organizations to identify revenue-optimizing opportunities instantly, avoiding the pitfall of "leaving money on the table" due to the rigidity of legacy infrastructure.

To ensure this value is delivered even during network instability, the terminal incorporates connectivity resilience via IndexedDB (Dexie.js), allowing it to function in a "degraded" offline mode and synchronize data the moment a connection to the local hub is restored.

By delivering a terminal interface that prioritises server-side execution, "Edge-Native" stability, and hardware-verified security, the project provides a scalable solution that ensures financial precision and operational excellence.

1.3 Scope and Technical Architecture

Wynsum POS Terminal, designed as the primary operational interface within an Edge-Native retail architecture. The system was strategically engineered as a robust, three-tiered stack to ensure scalability, data integrity, and high-concurrency performance under heavy retail loads.

1.3.1 Backend and Hub Logic (The "Single Source of Truth")

The backend, serving as the Wynsum Hub, was implemented using Python/Django with a modular API architecture. To ensure maximum performance for terminal operations, administrative functions were decoupled from sales workflows through two dedicated interfaces: `backoffice_api` and `pos_api`.

Asynchronous Checkout Pipeline: A central achievement of the backend development was the implementation of an asynchronous pipeline utilizing Django Channels and native async execution. This pipeline allows the Wynsum Hub to process complex, multi-step operations—including inventory updates, payment processing, and fiscalization—sequentially and asynchronously.

Terminal Responsiveness: This architecture ensures that the POS Terminal remains responsive without blocking the main execution thread, maintaining a local API response time of <5ms. All critical financial logic, such as recursive discounts and tax processing, was successfully extracted into a dedicated Service Layer to maintain a reliable "single source of truth" independent of the interface.

1.3.2 Frontend (The Wynsum POS User Experience)

The user interface was built as a high-performance Progressive Web App (PWA) using React, designed to meet the demands of fast-paced retail environments while remaining independent of specific hardware.

Universal Infinite List: To handle massive retail datasets, I developed a component employing row virtualization. This allows operators to browse through datasets exceeding 10,000 items (products, customers, or sales history) with a consistent performance of 60 FPS, significantly reducing memory overhead by recycling DOM nodes.

Connectivity Resilience: To ensure operational continuity during network instability, the frontend utilises IndexedDB (via Dexie.js) for local data persistence. This enables the POS terminal to function in a "degraded" offline mode, caching critical product data and using automated synchronisation logic to update the local hub instantly upon reconnection.

1.3.3 Communication and "Zero Trust" Terminal Security

The system employs a WebSocket architecture to facilitate a persistent, bi-directional connection between the Wynsum Hub and the POS hardware, enabling instantaneous price updates and real-time inventory synchronisation.

Zero Trust Model: Adhering to a "Zero Trust" security model, the project treats the frontend terminal as a potentially compromised environment where sensitive business logic should never be executed.

Cryptographic Handshake: To secure this layer, a cryptographic challenge-response protocol (ECDSA Device Verification) was integrated. This requires the POS terminal to sign server-issued challenges using non-exportable hardware keys, effectively verifying that only authorised, registered physical hardware can execute critical business logic. This approach prevents unauthorized access even in cases where user credentials might be compromised.

Note: This section was drafted by the author and refined for linguistic accuracy using Google Gemini and Grammarly (Google Gemini, 2026; Grammarly, 2026). See Appendix F for interaction logs.

1.4 Key Deliverables: The Wynsum POS Ecosystem

The development of the Wynsum POS Terminal and its supporting infrastructure comprises the following core deliverables, each engineered to facilitate high-speed retail operations while maintaining absolute data integrity:

1. "Thin-Client" POS Terminal Interface (React PWA): The primary frontend deliverable is a high-performance Progressive Web App (PWA) that serves as the operator's productivity workspace. Unlike legacy systems, this terminal is designed to be logic-light, delegating all critical financial computations to the local server. It features a Universal Infinite List with row virtualization to ensure a fluid 60 FPS experience during high-volume inventory searches and a local IndexedDB layer for connectivity resilience.

2. Server-Side Financial Logic (The "Single Source of Truth"): This central component, residing within the Wynsum Hub, is the foundation for the system's operational accuracy. It mandates the migration of all tax, transaction, and multi-layered financial calculations away from the POS terminal to the secure server. This deliverable ensures that the terminal always reflects data that is mathematically precise and compliant with regulatory standards, effectively establishing a unified "single source of truth" for the entire store.

3. High-Complexity Promotional Engine: A sophisticated, server-side mechanism engineered for the instantaneous calculation of intricate marketing rules. The POS terminal interacts with this engine in real-time to process:

Recursive Discounts: A specialized logic handler that manages multi-layered promotional rules applied sequentially. By executing this on the server, the terminal avoids the cumulative rounding errors and "infinite loop" scenarios typical of client-side legacy systems.

Dynamic Product Bundles: Real-time logic that allows the POS terminal to recognize specific product combinations (e.g., "Meal Deals") and automatically adjust the transaction total on the fly.

4. Optimized API Infrastructure (pos_api & backoffice_api): To support the POS terminal's high-concurrency requirements, the project delivered a segmented API architecture designed to eliminate traffic bottlenecks:

pos_api: The critical deliverable for terminal operations, built on a WebSocket architecture to provide a persistent, low-latency, bi-directional connection between the terminal hardware and the Hub.

backoffice_api: A dedicated RESTful interface for administrative management, reporting, and inventory configuration, ensuring that heavy management tasks do not interfere with the terminal's transaction speed.

5. Hardware-Verified Security Layer (Zero Trust Handshake): A robust security deliverable that implements ECDSA Device Verification. This ensures that the POS terminal interface can only execute business logic after a successful cryptographic challenge-response handshake. By verifying the terminal's physical identity through

non-exportable hardware keys, the system prevents unauthorized access even if software credentials are compromised.

1.5 Anticipated Technical Challenges

The implementation of the Wynsum POS Terminal and its integration into the broader Wynsum RMS ecosystem introduces a series of high-complexity technical challenges. Moving beyond basic CRUD operations, the system must navigate the intricacies of real-time state synchronization, cryptographic security, and transaction integrity in a high-pressure retail environment.

1. Protocol Consensus and Bi-Directional State Management

Unlike traditional HTTP-based applications, the Wynsum POS relies on a persistent WebSocket architecture to achieve a <5ms response time. The primary challenge lies in establishing "consensus" between the "thin" PWA client and the Wynsum Hub. Since WebSockets are inherently stream-oriented, I had to implement a custom JSON-RPC router to structure message exchange.

The Initial Handshake: Establishing a secure session is a multi-step process. The terminal must first perform a cryptographic handshake using ECDSA Device Verification, signing a server-issued challenge with a non-exportable hardware key.

Configuration Retrieval: Once verified, the POS must securely retrieve its specific operational context—including store-specific tax strategies, terminal permissions, and peripheral configurations—through a GET /meta or handshake request. Ensuring this configuration remains consistent across a persistent connection requires robust state management to handle potential "session hijacks" or silent disconnects.

2. Distributed Data Consistency and Delta-Synchronization

To ensure the terminal remains operational during network instability, the system employs IndexedDB (via Dexie.js) for local data persistence. However, maintaining a "Single Source of Truth" creates a significant versioning challenge:

Inventory Drift: The POS must ensure its local product cache is always current. I am implementing a delta-synchronization logic (handle_sync_items) that only fetches changes since the last known state, reducing payload size during peak operational periods.

Versional Integrity: The Client Installation Token (CIT) plays a critical role here, defining a maxAllowedVersion to ensure the PWA and the Hub remain compatible even after a local offline session.

3. Asynchronous Transaction Integrity (The Order Pipeline)

Implementing a reliable checkout process without a traditional "heavy" client requires a sophisticated State Machine on the server side to avoid "Transaction Hell".

Soft Reserve Mechanism: When a cashier initiates a payment, the system must immediately trigger a "Soft Reserve" on the Hub, locking inventory items to prevent over-selling before the payment is finalized.

The Pipeline Workflow: The terminal must orchestrate a complex, multi-step Asynchronous Checkout Pipeline that handles inventory updates, external payment gateway callbacks (e.g., Stripe, Wero), and fiscalisation sequentially without blocking the main UI thread. Managing this workflow over WebSockets is technically demanding, as the terminal must gracefully handle "network drops" mid-transaction and resume the specific state from the Hub once reconnected.

4. Mathematical Precision in Recursive Financial Logic

Migrating the Promotional Engine to the server side solves the problem of "leaving money on the table" through rounding errors, but it introduces the challenge of recursive complexity.

Loop Prevention: Implementing recursive discounts that apply multi-layered promotional rules sequentially requires a priority-based execution stack. Without precise engineering, such logic can fall into "infinite loop" scenarios, especially when handling dynamic product bundles in real-time. The challenge is ensuring these complex calculations execute within the operational transaction cycle without introducing latency at the point of service.

Research and Analysis

The development of Wynsum RMS is predicated on addressing the structural failure of both legacy local systems and modern cloud-native platforms, a phenomenon identified as the "Hybrid Paradox". While legacy systems like RMH or Microsoft Dynamics RMS offer offline reliability, they suffer from architectural rigidity; conversely, cloud systems like Lightspeed introduce risks of internet dependency and forced processing fees.

To bridge this gap, the core requirements for the Wynsum POS Terminal and its supporting backend are defined as follows:

2.1 Requirements Analysis

The functional requirements are specifically designed to resolve the "operational disconnect" where sophisticated marketing needs outpace the technical capabilities of existing retail infrastructure.

1. Recursive Discount Logic

Legacy Limitation: Industry standards, such as RMH, often rely on client-side logic for financial calculations. This introduces a high risk of rounding errors and conflicting discount rules, effectively "leaving money on the table" for the retailer.

Wynsum Requirement: The platform requires a sophisticated, server-side Promotional Engine capable of handling multi-layered promotional rules applied sequentially. By migrating this logic to the Wynsum Hub, the system establishes a reliable "single source of truth," ensuring mathematical precision and preventing price manipulation risks associated with traditional client-side methods.

2. Dynamic Product Bundles

Legacy Limitation: Traditional systems often struggle to measure the true financial impact of complex pricing strategies or lack the agility to recognize bundled deals in real-time.

Wynsum Requirement: The system must provide real-time recognition of specific product combinations (e.g., "Meal Deals"). The Promotional Engine is required to automatically adjust the total price at the POS terminal instantaneously, allowing organizations to maximize "total guest value" and optimize revenue through a unified operational cycle.

3. Centralised Tax Processing

Legacy Limitation: Discrepancies between the terminal display and the final transaction record frequently occur in systems where logic is fragmented across multiple client nodes. Furthermore, storing sensitive data in plaintext within legacy SQL databases poses a significant GDPR risk.

Wynsum Requirement: To ensure absolute financial compliance and data integrity, all tax calculations must be mandated as a server-side process. This ensures that the POS

terminal functions as a "thin" interface while the local server (Wynsum Hub) handles the regulatory-heavy business logic. This centralized approach ensures mathematical precision even under heavy load and aligns with "privacy by design" principles by securing data within an encrypted local environment.

By implementing these requirements through an Edge-Native architecture, Wynsum RMS provides the speed of a local API (<5ms) with the intelligence of a modern cloud platform, specifically tailored for the high-concurrency needs of the European retail market.

2.2 Technical Research

The development of the Wynsum POS Terminal and its supporting ecosystem is based on a rigorous technology stack selected to meet the high-concurrency demands of modern European retail environments. This research-led approach prioritises architectural stability, low-latency responsiveness, and absolute data integrity.

Python/Django (Backend & Hub Logic): Django was selected as the core framework for the Wynsum Hub due to its enterprise-grade efficiency in managing complex, multi-layered relational data models. To ensure maximum operational performance, the project employs a split-API strategy, decoupling administrative management from terminal transactions through two dedicated interfaces: `backoffice_api` (REST-based) and `pos_api` (real-time WebSocket-based).

A critical architectural achievement was the implementation of an Asynchronous Checkout Pipeline using Django Channels and native async execution. This allows the Hub to process multi-step tasks—such as inventory updates, payment callbacks (including future Wero A2A integrations), and fiscalization—sequentially and asynchronously, ensuring the POS terminal remains responsive even under peak transaction loads. Furthermore, a "Fat Models/Services" philosophy was adopted, moving critical financial logic into a dedicated Service Layer to establish a reliable "single source of truth" independent of the frontend.

React (Frontend & PWA): The POS terminal is engineered as a high-performance Progressive Web App (PWA) built with React. This approach allows for a hardware-agnostic, iPhone-like user experience (UX) with the reliability of a local system. Technical research into the Virtual DOM confirms its superiority in maintaining UI responsiveness during rapid, complex transaction updates.

To address the challenge of massive retail datasets, a custom Universal Infinite List component was developed using row virtualization. This allows operators to browse through 10,000+ items (products, sales history, or customers) with a consistent performance of 60 FPS while significantly reducing browser memory overhead.

WebSocket Architecture & JSON-RPC: Research into latency reduction confirms that WebSockets are vastly superior to traditional HTTP polling for retail environments (Forward, 2015). The system maintains a persistent, bi-directional connection between the Wynsum Hub and all connected POS terminals, facilitated through Daphne/Uvicorn and Redis.

This allows for instantaneous price updates and real-time inventory synchronisation across the store. To structure this communication, a custom JSON-RPC router was implemented, providing a robust protocol for managing method calls such as `calculate_cart` and `login_cashier`.

Local Persistence & Connectivity Resilience: To achieve "Edge-Native" stability, the terminal utilises IndexedDB (via Dexie.js) for local data persistence. This ensures that even during total network provider failure, the POS remains 100% functional, caching scans and using delta-synchronization logic to update the local Hub immediately upon reconnection.

"Zero Trust" Security Layer: Adhering to modern cybersecurity standards, the architecture treats the terminal as a potentially compromised environment. Security research led to the integration of a cryptographic challenge-response protocol based on ECDSA Device Verification.

This requires the POS hardware to sign server-issued challenges using non-exportable keys, ensuring that only verified, registered physical devices can execute critical business logic on the Hub. This protects the business against common retail vulnerabilities like RAM-scraping and unauthorised API access.

2.3 Market Analysis: Wynsum RMS's Unique Value Proposition

A modern Retail Management System (RMS) is required to evolve beyond a simple Point of Sale (POS) terminal, serving concurrently as a compliant "system-of-record" and an intelligent, high-performance "productivity workspace". Current market analysis of the Irish and European retail landscape reveals a "Hybrid Paradox": legacy systems (e.g., RMH, CBE) offer offline reliability but suffer from architectural rigidity, while cloud-native platforms (e.g., Lightspeed, Shopify) offer mobility but introduce critical connectivity risks and forced transaction fees.

Specifically, existing competitors suffer from significant structural limitations:

Connectivity Fragility and "Pseudo-Offline" Modes: Cloud-based systems rely on the public internet for core business logic. Their "offline modes" are technically fragile, relying on browser caching (LocalStorage) that often blocks critical functions—such as real-time inventory checks, new customer creation, or integrated card processing—during an ISP outage.

Operational Disconnect via Client-Side Logic: Many industry-standard systems rely on client-side terminal logic for financial calculations. This introduces a high risk of rounding precision errors, conflicting promotional rules, and manual reprogramming inefficiencies, effectively "leaving money on the table" for the retailer.

Payment Processing Monopolies: Global SaaS providers increasingly impose "forced payments" policies, charging significant penalties or higher subscription rates to merchants who prefer to use their local Irish banks (AIB, BOI). This limits the financial autonomy of the retailer.

Wynsum RMS distinguishes itself through three unique value propositions:

"Edge-Native" Stability with Server-Side Data Integrity: Unlike cloud-only systems, Wynsum utilizes a "Wynsum Hub" (a local, industrial-grade Private Cloud server) to host the pos_api and all critical business logic. By migrating recursive discounts, tax processing, and inventory management to the local server side, the project establishes a robust "single source of truth". This architecture delivers the reliability of a legacy system with the intelligence of a cloud platform, providing ultra-low API response times (<5ms) and 100% operational functionality even during total internet failure.

Focus on Total Guest Value and Revenue Optimization: While many market products focus solely on basic transaction processing, the Wynsum POS Terminal is engineered as a revenue optimization tool. The server-side Promotional Engine is designed to accurately measure the true financial impact of complex pricing strategies, such as multi-layered loyalty rewards or dynamic product bundles, ensuring that complex business logic does not create bottlenecks during peak operational periods.

Payment Autonomy and Future-Proof Connectivity: Wynsum is fundamentally "Payment Agnostic". Through its modular "Payment Provider Layer," the system empowers retailers to integrate any local payment provider or future A2A (Account-to-Account) standards like Wero without incurring penalty fees. By providing hardware-verified security through a "Zero

Trust" tunnel, Wynsum ensures that marketing and financial strategies do not inadvertently limit operational growth, while remaining "Grant-Ready" for Irish government support programs (e.g., LEO Digitalisation Vouchers).

3. Modelling and Design

This section present the structure of the DB and the system design

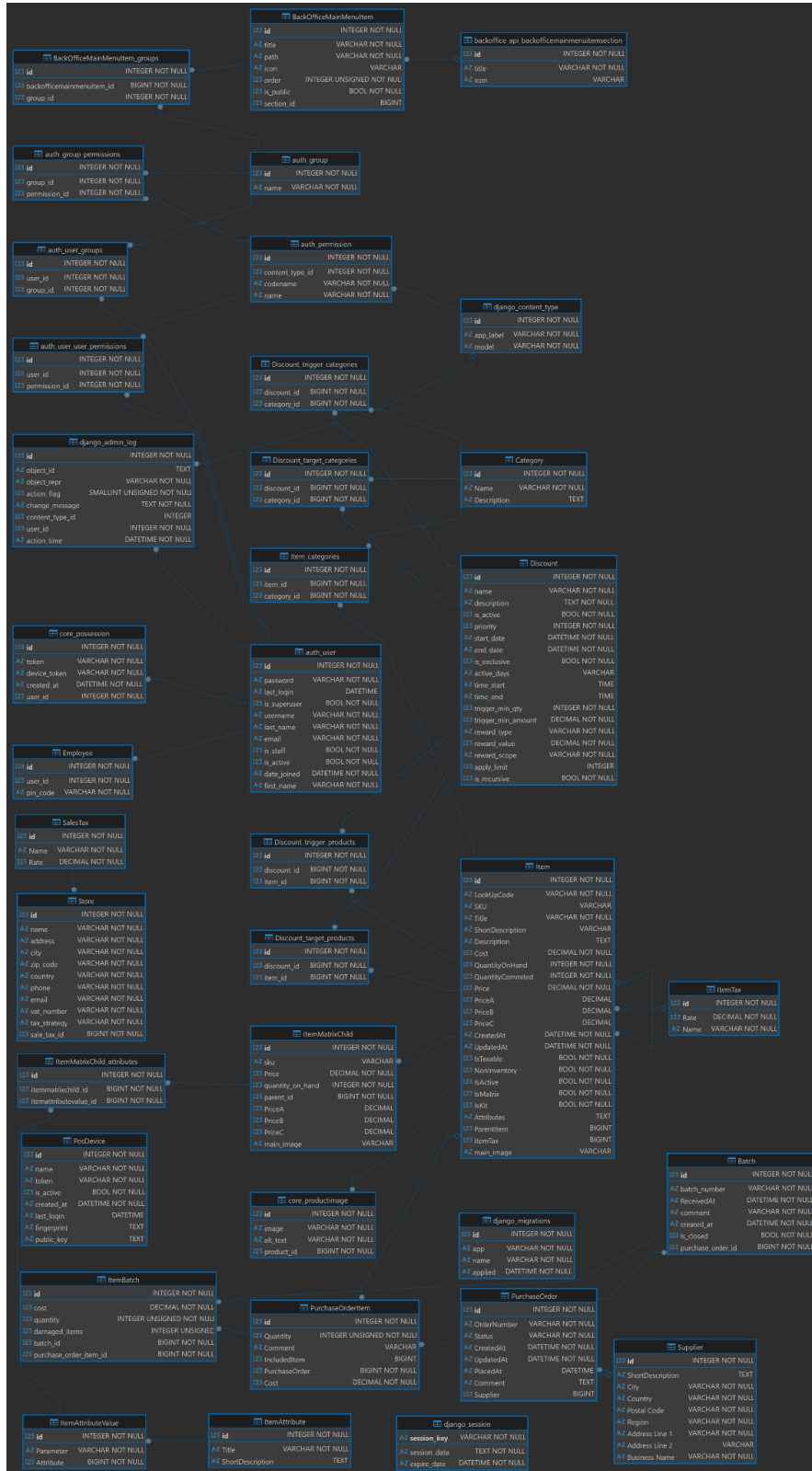


Figure 1: Entity Relationship Diagram of Wynsum Hub

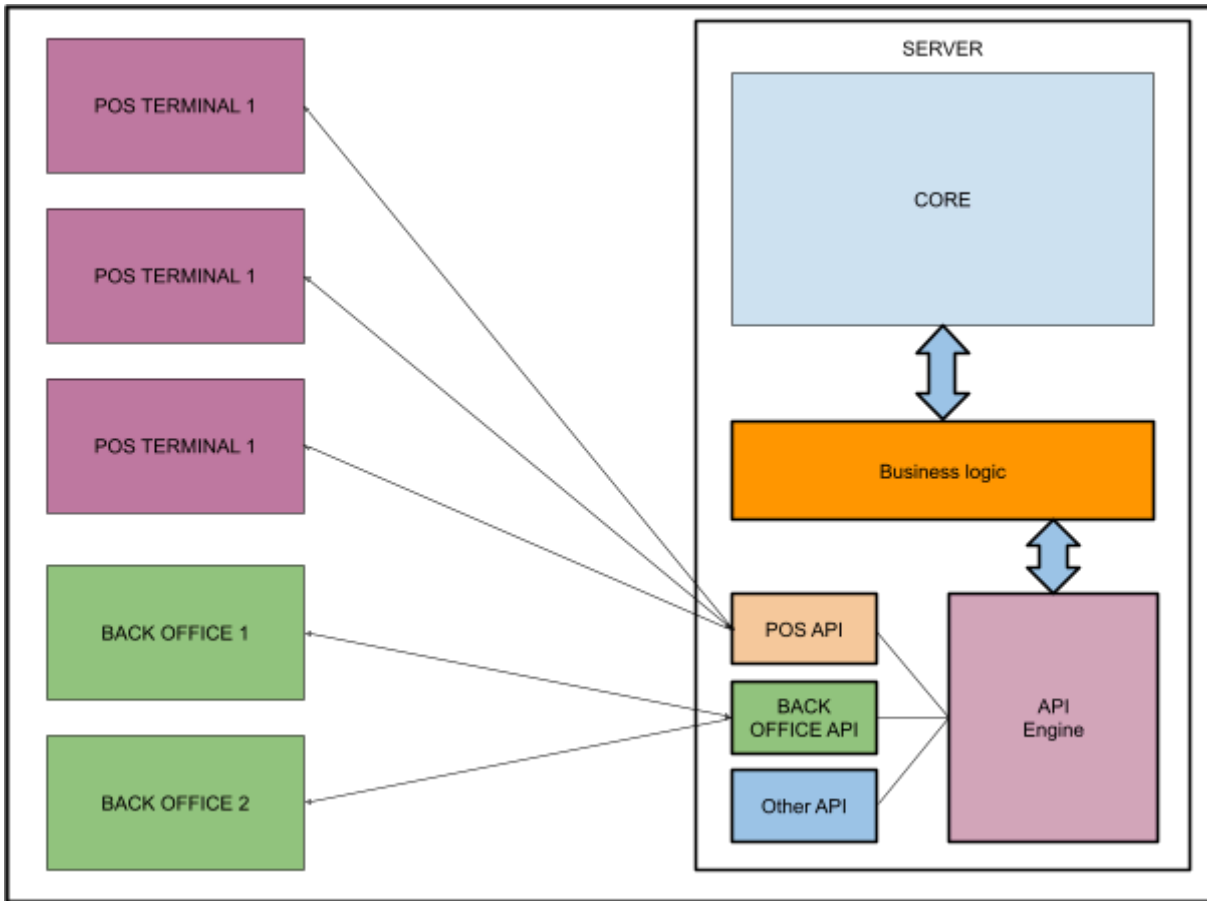


Figure 2: Edge-Native System Architecture Overview

Project Management

The development of Wynsum RMS is governed by a rigorous project management framework to ensure that all critical milestones are met on time, within scope, and adhere to a high standard of quality.

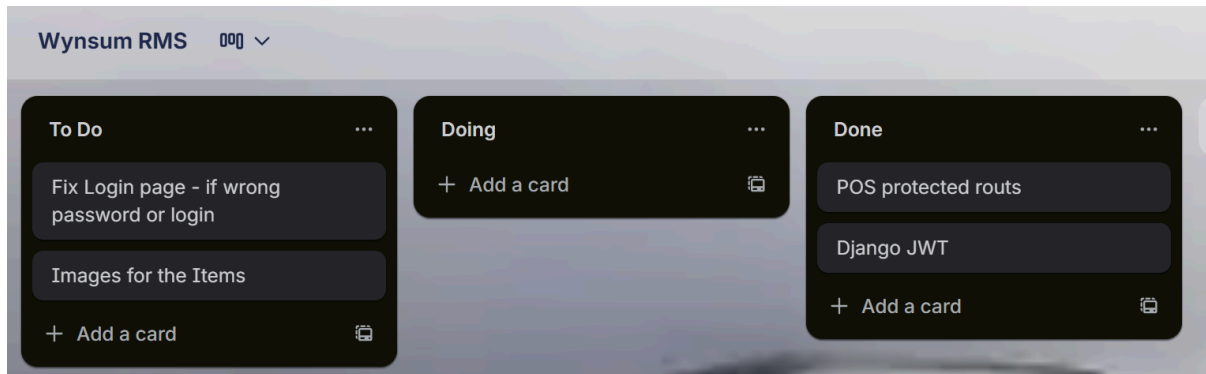


Figure 3: Trello board example

N	Task / Stage	Started	Adding Logic	Testing	Committed	Progress
						30.357
1	Checkout Logic	1	1		1	75
2	Login Issue with no network					0
3	Orders List	1	1		1	75
4	Items List	1	1		1	75
5	Add server search screen/functionality	1	1		1	75
6	Print recieprs	1	1		1	75
7	Open shift					0
8	Close shift					0
9	POS settings get from the backend	1	1			50
10	Single order view					0
11	single item view					0
12	items returns					0
13	загрузка валюти розрахунків для відображення у списках ордерів і будь де					0
14	customer display for a price etc					0
15	Add manual discount					0
16	POS installation from the domain					0
17						
18						
19						
20						

Figure 4: Custom task tracking list using MS Excel

4.1 Methodology: Agile/Scrum Framework

The project adopts the Agile/Scrum methodology. This approach is specifically chosen to manage the complexity of full-stack development, ensuring flexibility, rapid iterative progress, and effective risk reduction through continuous feedback loops. The project is systematically divided into two-week iterations, allowing for regular assessment and adjustment of features based on technical progress and requirements.

- **Backlog Management:** Core system requirements, such as the server-side recursive discount logic, are formalised as User Stories. These are prioritised in a product backlog which drives the content of each sprint.
- **Continuous Integration:** The system employs GitHub for robust source code management. Daily commits are mandated to maintain a transparent development history.
- **Communication:** Technical discussions and project health checks are managed through a dedicated Slack channel, ensuring real-time contact with the project supervisor.

4.2 Sprint Plan and Phasing

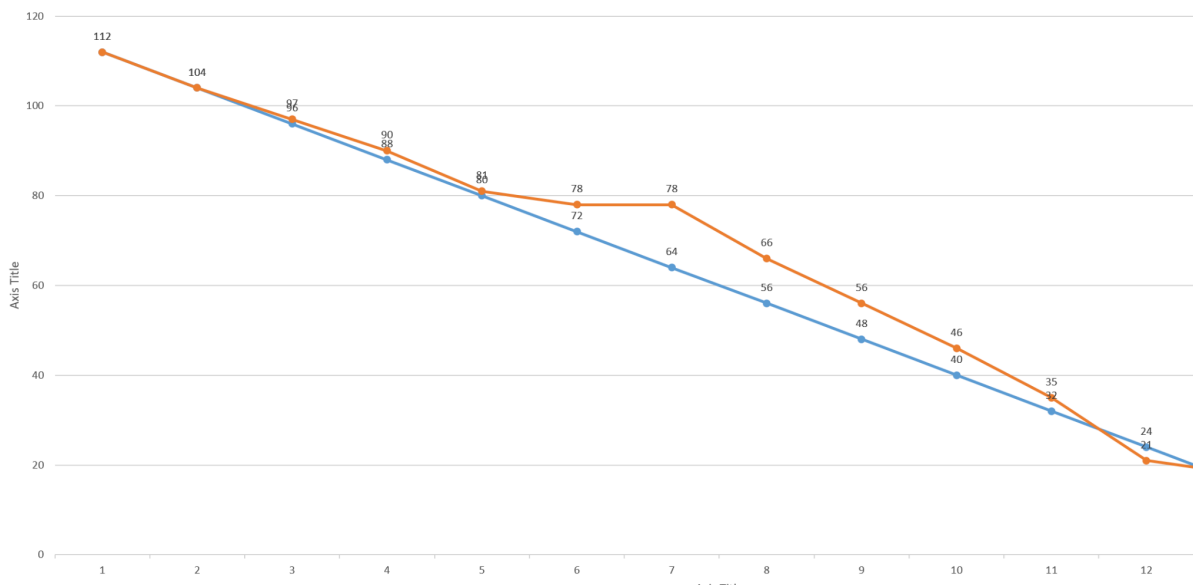


Figure 5: Scrum 1 Burndown Chart demonstrating sprint velocity

The development lifecycle of the Wynsum POS Terminal was executed across seven distinct, two-week sprints. The project was managed using an Agile/Scrum framework, with a Trello Kanban board serving as the central hub for tracking user stories, prioritising the product backlog, and verifying the "Definition of Done" for each feature.

Phase 1: RAMP and Architectural Design (Sprints 1–2)

The initial phase was dedicated to foundational research and high-level system modelling. During this period, the Entity-Relationship (ER) diagram was finalised to support complex retail data structures, including recursive discounts and multi-layered tax strategies.

A critical architectural decision was made to decouple the system into a segmented API architecture: a REST-based `backoffice_api` for administrative management and a high-performance, WebSocket-ready `pos_api` for terminal operations. This phase also included the design of the "Zero Trust" security model, involving research into ECDSA Device Verification to ensure that only authorised hardware could interface with the Hub.

Phase 2: Core Implementation of the POS Ecosystem (Sprints 3–5)

This central phase represented the most intensive period of technical development, focusing on the core functional deliverables of the project:

Implementation of the Asynchronous Checkout Pipeline (Scrum 1): Following the requirements for high-concurrency performance, a robust checkout process was engineered using Django Channels and native async execution. This pipeline allowed the Hub to handle multiple sequential steps—Inventory Reservation, Payment Processing, and Fiscalization—asynchronously, ensuring the terminal interface remained responsive even while awaiting external gateway callbacks. Functionality for Split Payments (mixed card and cash) and automated change calculation was integrated directly into the server-side logic to maintain a "single source of truth".

High-Performance UI with Virtual Scrolling (Scrum 2): To meet the demands of modern retail, I developed a Universal Infinite List component for the React frontend. By implementing row virtualisation, the terminal achieved a consistent performance of 60 FPS while browsing massive datasets exceeding 10,000 items, such as product catalogues and customer records.

The Server-Side Promotional Engine: A sophisticated mechanism was implemented to process intricate marketing rules. This included the Recursive Discount Logic, which utilised a priority-based execution stack to calculate multi-layered promotions in real-time without falling into "infinite loop" scenarios.

Phase 3: Refinement, Testing, and Dissemination (Sprints 6–7)

The final phase focused on system stability and the delivery of dissemination materials as required by the programme guidelines.

WebSocket Integration and Latency Optimization: The system transitioned from HTTP polling to a persistent WebSocket architecture, achieving the goal of a <5ms local API response time for real-time inventory and price updates.

Resilience and Data Consistency: Rigorous testing was conducted on the connectivity resilience features. This verified that the IndexedDB (Dexie.js) local persistence layer correctly handled "degraded" offline sessions and performed seamless delta-synchronisation upon reconnection to the Hub.

Final Deliverables: The project culminated in the production of a 7-10 minute demonstration video highlighting the cryptographic handshake, the finalisation of the Project Landing Page, and the preparation of the Showcase Entry materials for the SETU Computing Expo.

The Scrum 3 example:

Sprint Plan: Dynamic Keypad System

Sprint Goal: Implement a high-performance, real-time configurable Keypad System for the POS terminal to handle dynamic layouts, folders (menus), and quick actions with seamless server-side synchronization via WebSockets.

Duration: 2 weeks

Tech Stack: Python (Django + Channels), React (TypeScript + MUI), PostgreSQL, Dexie (IndexedDB).

Epic 1: Dynamic Layout & Grid (Frontend)

User Story 1.1: As a cashier, I want to see a grid of buttons (Products/Actions) so that I can quickly process sales without searching the full database.

Priority: Critical (High)

Tasks:

- [Frontend] Implement KeypadGrid: Create a responsive container that renders MenuButton components based on a 1-level deep hierarchy.
- [Frontend] Button Styling Logic: Implement dynamic background colors and automatic text contrast (black/white) using MUI getContrastText.
- [Frontend] Action Dispatcher: Create a handler for ADD_PRODUCT, OPEN_MENU, and CUSTOM_ACTION types.

User Story 1.2: As a cashier, I want to navigate through folders (Menus) and back to the main screen so that I can access categorized items (e.g., Drinks, Desserts).

Priority: High

Tasks:

- [Frontend] Navigation State: Implement a currentMenuId state to filter and display only relevant buttons in the viewport.
- [Frontend] Breadcrumbs/Back Button: Add a navigation element to return from a sub-menu to the Layout Root.
- [Frontend] Empty States: Design a "Staging Area" view for layouts with no assigned buttons.

Epic 2: Real-time Sync & Persistence (Fullstack)

User Story 2.1: As a manager, I want changes made in the admin panel to appear instantly on the POS screen so that I can update prices or buttons during a shift.

Priority: High

Tasks:

- [Backend] Unified Layout Signal: Implement a Django Signal observer for KeypadLayout, QuickMenu, and QuickButton to trigger WebSocket broadcasts.
- [Backend] Layout Serializer: Create a high-performance KeypadLayoutDetailedPOSSerializer with optimized prefetch_related for buttons and menus.
- [Frontend] WebSocket Listener: Implement a keypad_layout_update effect to intercept server notifications and refresh the UI state.

User Story 2.2: As a system, I must store the layout in a local database (IndexedDB) so that the keypad remains functional even after a page refresh or temporary network loss.

Priority: High

Tasks:

- [Frontend] Dexie Integration: Implement a "Get-Modify-Put" logic to update the keypad_layout field within the pos_settings record in IndexedDB.
- [Frontend] Initial Boot Logic: Ensure the system loads the layout from LocalDB first, then hydrates it with fresh data from the RPC call.

Epic 3: Integrity & Validation (Backend)

User Story 3.1: As a developer, I want to enforce a strict 1-level nesting limit to keep the UI simple and prevent infinite loops in navigation.

Priority: Medium

Tasks:

- [Backend] Strict Validation Logic: Implement a clean() method in QuickButton to block folder-type buttons from being placed inside other folders.
- [Backend] Cross-Layout Protection: Ensure buttons cannot link to menus belonging to a different KeypadLayout.

User Story 3.2: As an administrator, I want to manage layouts through a clean interface without creating redundant "Root" entities.

4.3 Project Completion and Commercial Evolution

By the conclusion of the final development cycle in late March 2026, the project successfully transitioned from the implementation phase to a feature-complete state. All seven sprints were executed according to the Agile/Scrum framework, resulting in a robust, edge-native infrastructure.

The following core milestones have been fully achieved:

- **Production-Ready Django Architecture:** The backend environment is fully configured with a decoupled, modular structure (pos_api and backoffice_api), supporting high-concurrency retail loads.
- **Asynchronous Checkout Pipeline:** A functional state-machine-based order pipeline was implemented, utilizing Django Channels for non-blocking multi-step transactions.
- **Server-Side Promotional Engine:** The central logic for recursive discounts and dynamic product bundles was successfully migrated to the server-side Service Layer, establishing a reliable "single source of truth".
- **High-Performance Frontend:** The React/Vite PWA is fully operational, featuring a Universal Infinite List with row virtualization capable of managing over 10,000 items at a consistent 60 FPS.
- **Zero Trust Security Layer:** The system successfully integrates ECDSA hardware verification, ensuring that only authenticated physical devices can interface with the Wynsum Hub.

While the academic requirements of this project have been satisfied, **Wynsum RMS is a commercial-grade platform**. As such, the project is now evolving beyond the initial development scope. Work continues toward a production-grade deployment, including migration to hardened "Wynsum Hub" hardware and full containerization via Docker Compose to ensure 24/7 operational resilience. The system has moved from a technical demonstration to a market-ready solution capable of addressing the complex needs of modern retail infrastructure.

Note: This section was drafted by the author and refined for linguistic accuracy using Google Gemini and Grammarly (Google Gemini, 2026; Grammarly, 2026). See Appendix F for interaction logs.

Current Status and Implementation Outcomes

This section details the functional maturity of the Wynsum POS Terminal as of the conclusion of the core development phase. It outlines the high-complexity technical milestones that were successfully achieved, establishing the terminal as a robust, "thin-client" interface integrated within an Edge-Native retail environment.

5.1 Implementation Status

By the reporting period, the Wynsum POS had reached a significant level of operational maturity. The core architecture was successfully established, strictly adhering to the "Zero Trust" and "Edge-Native" principles defined during the RAMP phase.

Wynsum Hub Infrastructure (Backend): The Django-based local server was made fully operational. It utilized a modular structure that successfully decoupled administrative functions from high-speed terminal operations through the dedicated `pos_api`. A major achievement was the full integration of the Asynchronous Checkout Pipeline, utilizing Django Channels and WebSockets to ensure the POS terminal remained responsive during complex multi-step transactions.

Database Integration and Entity Modeling: The relational schema was successfully implemented and optimized for retail-specific data. The database currently supports complex item matrices, multi-layered tax strategies, and recursive discount configurations, providing a stable foundation for the "Single Source of Truth".

Server-Side Promotional Engine: The core logic for calculating recursive discounts and dynamic product bundles was successfully migrated to the server-side Service Layer. This refactoring ensured that the POS terminal no longer handles financial calculations, thereby eliminating the rounding errors and price manipulation risks inherent in legacy client-side systems.

POS Terminal Interface (PWA): The frontend was developed as a high-performance Progressive Web App (PWA) using React and Vite. The terminal was engineered as a hardware-agnostic "productivity workspace," featuring a Universal Infinite List with row virtualization. This allowed for the fluid browsing of over 10,000 items at a consistent 60 FPS, ensuring an industrial-grade user experience.

Connectivity Resilience and Local Persistence: A robust local persistence layer utilizing IndexedDB (Dexie.js) was fully implemented. This enabled the POS terminal to function in a "degraded" offline mode during network instability, with a custom delta-synchronization logic developed to update the local Hub instantly upon reconnection.

Cryptographic "Zero Trust" Handshake: The security layer was completed with the implementation of an ECDSA-based challenge-response protocol. This verified the authenticity of the physical POS hardware, ensuring that only registered devices could initiate sessions or execute critical business logic on the Hub.

By achieving these milestones, the project has delivered a scalable, secure, and mathematically precise POS solution that addresses the critical failures of legacy retail infrastructure.

5.2 Problems Encountered and Resolutions

During the development of the Wynsum POS Terminal, the project moved beyond theoretical design to address several high-complexity engineering hurdles. The following real-world challenges were successfully resolved:

Establishing Protocol Consensus over Stream-Oriented WebSockets: One of the most significant technical hurdles was transitioning from stateless HTTP to a persistent, bi-directional WebSocket connection. Because raw WebSockets are inherently stream-oriented, achieving a structured "request-response" consensus between the POS terminal and the Hub was difficult.

- Resolution: I developed a custom JSON-RPC router (`rpc_utils.py`) to standardize message exchange. This allowed the terminal to execute specific remote methods (such as `calculate_cart` or `login_cashier`) and receive typed responses, effectively creating a reliable communication protocol that handles high-concurrency traffic without message collision or data corruption.

Synchronizing Operational Context and Inventory Parity: Ensuring the POS terminal possessed the correct operational settings (tax strategies, terminal permissions) upon connection was a major challenge. There was a high risk of "Inventory Drift," where the local terminal's database could fall out of sync with the Hub's "Single Source of Truth," especially after an offline session.

- Resolution: This was resolved by implementing a Client Installation Token (CIT) and a specialized handshake/meta retrieval workflow. Upon connection, the terminal performs a cryptographic handshake to verify its identity. Once verified, it executes a delta-synchronization logic that updates the local IndexedDB only with changes made since the last known version, ensuring the terminal always operates with the most current product data and configurations.

Ensuring Transaction Integrity via the "Order Pipeline" State Machine: Managing the transition from a cart to a finalized sale introduced the risk of "Transaction Hell"—a scenario where a network drop during a multi-step checkout could leave inventory locked or orders in an ambiguous state.

- Resolution: I implemented a robust State Machine-based Order Pipeline (Draft → Created → Paid) on the server side. By introducing a "Soft Reserve" mechanism, inventory is temporarily locked on the Hub once a payment is initiated. If the connection is lost, the Hub maintains the current state, allowing the POS terminal to resume the specific transaction workflow exactly where it left off upon reconnection, thereby protecting the integrity of the warehouse data.

Architectural Refactoring for Service-Oriented Modularity: Initially, a significant amount of complex retail logic was embedded directly within the WebSocket Consumers, making the system difficult to maintain and test.

- Resolution: I conducted a major architectural refactoring to adhere to the "Fat Models/Services, Thin Views" philosophy. All critical business rules—including the Promotional Engine and cart calculations—were extracted into a dedicated Service Layer (`core/services/sells.py`). This refactoring reduced the WebSocket consumer to a clean interface adapter, ensuring that the core business logic remains independent of the transport layer and can be rigorously tested in isolation.

5.3 Future Work: Transitioning to Production-Grade Infrastructure

Moving forward, the project will shift focus from feature development to high-availability deployment and long-term operational stability. The following tasks are scheduled for completion to bring the system to a commercial production standard:

Secure PWA Hosting and Authorized Deployment: The POS terminal will be transitioned to a secure production domain. To prevent unauthorized access to the proprietary interface, the deployment will utilize a Client Installation Token (CIT) system.

This mechanism ensures that the PWA can only be installed via unique, signed installation links provided to authorized clients, effectively securing the frontend within a "Zero Trust" framework.

Migration to the "Wynsum Hub" (Hardened Appliance): The backend will be migrated from a development environment to a dedicated industrial Mini-PC (Wynsum Hub), featuring an AMD Ryzen 7 architecture to maintain ultra-low API response times of <5ms.

This migration involves more than a data transfer; it requires hardening the environment using Headless Linux (e.g., Ubuntu Core) and implementing LUKS disk encryption to ensure GDPR compliance for data-at-rest.

DevOps and System Resilience: To ensure the 24/7 uninterrupted operation required in retail, the system will be fully containerized using Docker Compose.

Each component—including the Django Core, Redis message broker, and PostgreSQL database—will be configured with strict resource limits and automated restart: always policies.

This Microkernel approach ensures that even if a non-critical "satellite" module fails, the core sales functionality of the POS remains operational.

WebSocket and JSON-RPC Finalization: The project will complete the transition to a fully persistent, bi-directional WebSocket architecture.

This includes the final refinement of the custom JSON-RPC router to manage real-time inventory synchronization and instantaneous price updates across all connected terminals.

End-to-End Stress Testing and Expo Preparation: Rigorous stress testing of the server-side Promotional Engine will be conducted.

This involves simulating thousands of high-concurrency transaction combinations to verify that the recursive discount logic maintains mathematical precision without introducing latency. Finally, work will commence on dissemination materials for the SETU Computing Expo, including a 7–10 minute video demonstrating the system's cryptographic handshake and offline resilience.

By implementing these production-level DevOps strategies, Wynsum RMS will transition from a technical demonstration to a market-ready solution capable of challenging established legacy providers.

5.4 What I Learned: Architectural DevOps Upskilling

Throughout the development of the Wynsum RMS ecosystem, my primary learning curve shifted from feature-oriented coding to high-level architectural orchestration. Managing a project of this scale and complexity required a transition from basic full-stack development to what can be described as **Architectural DevOps engineering**.

The following key areas represent the core professional growth I achieved:

1. Architectural Decoupling and Systemic Modularity I learned the critical importance of a **decoupled API architecture** in high-concurrency environments. By splitting the system into `backoffice_api` and a WebSocket-based `pos_api`, I successfully separated heavy administrative tasks from time-critical sales workflows. This taught me how to design systems that maintain operational velocity under load, ensuring the POS terminal remains responsive (<5ms) regardless of backend management activities.

2. Distributed Data Consistency and Consensus Moving beyond traditional CRUD operations, I gained deep insights into **Distributed Systems Theory**. Implementing connectivity resilience via IndexedDB and Dexie.js required a sophisticated understanding of delta-synchronization and versional integrity. Learning how to establish a "Single Source of Truth" while allowing local nodes to operate in degraded offline modes was a major leap in my ability to design resilient industrial software.

3. The "Zero Trust" Security Mandate One of the most significant architectural lessons was the implementation of a **Zero Trust security model**. I learned that security is not just an add-on but a fundamental architectural pillar. Mastering the integration of **ECDSA cryptographic verification** taught me how to treat a frontend environment as potentially compromised and move all business-critical logic exclusively to a hardware-verified secure layer.

4. Production-Ready DevOps and Resilience The project served as an intensive "DevOps upskill." I learned to look beyond the code toward **operational stability**. This involved mastering containerization via **Docker Compose** for microkernel-style resilience and researching hardware-level hardening (LUKS encryption, Ubuntu Core) for the Wynsum Hub

Mini-PC deployment. This experience taught me how to bridge the gap between a "working application" and a "market-ready solution" capable of 24/7 retail operation.

5. Engineering Consensus over Persistent Streams Developing a custom **JSON-RPC router** to manage bi-directional communication over WebSockets was a pivotal learning experience. It taught me how to structure request-response consensus in a stream-oriented environment, ensuring that high-concurrency retail traffic is handled without message collision or data corruption.

Note: This section was drafted by the author and refined for linguistic accuracy using Google Gemini and Grammarly (Google Gemini, 2026; Grammarly, 2026). See Appendix F for interaction logs.

AI Disclosure and Methodology

This ~8,000-word report was developed using a hybrid approach involving original research, software development, and AI-assisted structuring and editing. As a non-native English speaker, I utilized these tools to ensure the report meets professional academic standards while maintaining the originality of the core concepts and analysis.

1. Strategic Research and Technical Benchmarking

During the initial design phase of the **Wynsum™ RMS** ecosystem, I utilized **Google Gemini's Deep Research** capabilities. This was instrumental in:

- Conducting a comprehensive **competitor analysis** within the POS and Retail Management sector.
- Evaluating various **technical architectures** to ensure the scalability and security of the Wynsum POS Terminal.
- Refining the **business value proposition** of the project. While the insights gained from this research informed the project's strategic direction and architectural decisions, the research was used as a conceptual foundation rather than for direct text generation.

2. Structural Planning and Alignment

Google NotebookLM was used to process the official project briefings and documentation. The tool assisted in generating a high-level report structure with brief introductory prompts for each section, ensuring the report fully aligns with the project's specific learning outcomes.

3. Linguistic and Grammatical Refinement

To ensure academic clarity as a non-native speaker:

- **Grammarly** was used for real-time grammar, syntax, and stylistic corrections.
- **Google Gemini** assisted in translating and rephrasing complex technical arguments originally drafted in Ukrainian, ensuring the nuances of the software logic were accurately conveyed in professional English.
- The interaction logs provided in Appendix F represent the primary drafting and conceptualization phases. Due to the iterative nature of the writing process, many sections underwent multiple manual revisions and further linguistic refinements where logs may be incomplete. However, the provided examples accurately reflect the methodology of AI assistance used throughout the project.

4. Software Development and Implementation

- **Original Logic:** All core software logic, database schemas, and the specific implementation of the **Wynsum POS Terminal** are the original work of the author.
- **Code Documentation:** AI tools were utilized to automate the generation of code comments and docstrings. This was done to ensure the source code remains clear, professional, and follows industry-standard documentation practices.

- **Integrity:** All AI-generated suggestions (textual and technical) were critically reviewed and manually integrated to ensure they accurately reflect the project's unique requirements.
- **Commercial Intent:** As this project is a commercial venture with a registered trademark (**Wynsum™**), AI tools were employed to ensure that the documentation reflects professional industry standards.

5. Section: Copyright and Intellectual Property

- **Tool:** Google Gemini
- **Usage:** The entire text of the 'Statement of Copyright and Intellectual Property' was generated by the AI based on my specific requirements regarding the ownership of the Wynsum™ RMS ecosystem. This was done to ensure the terminology meets professional standards, as I am not a legal expert.

Appendix E

Feedback Report

No human experimentation issues (UG).

No animal experimentation issues (N/A).

No issues regarding the use of human tissues.

No animal tissue or biological fluids issues.

No ionising radiation issues.

No primary data collection issues (N/A).

No underage/vulnerable people issues (UG).

No issues regarding existing/secondary data use (N/A).

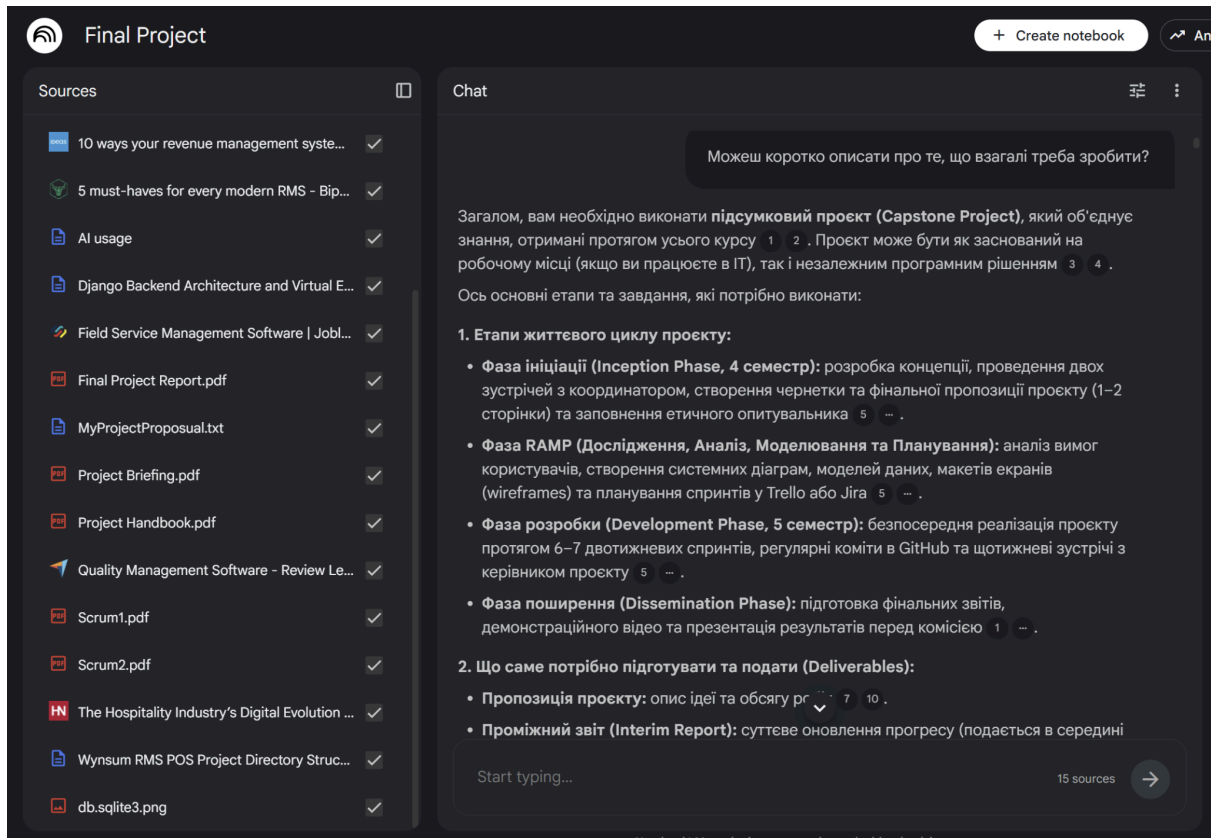
No controversial data issues.

No issues related to the collection of rare or protected plants.

No issues regarding the use of genetically modified (GM) plant material.

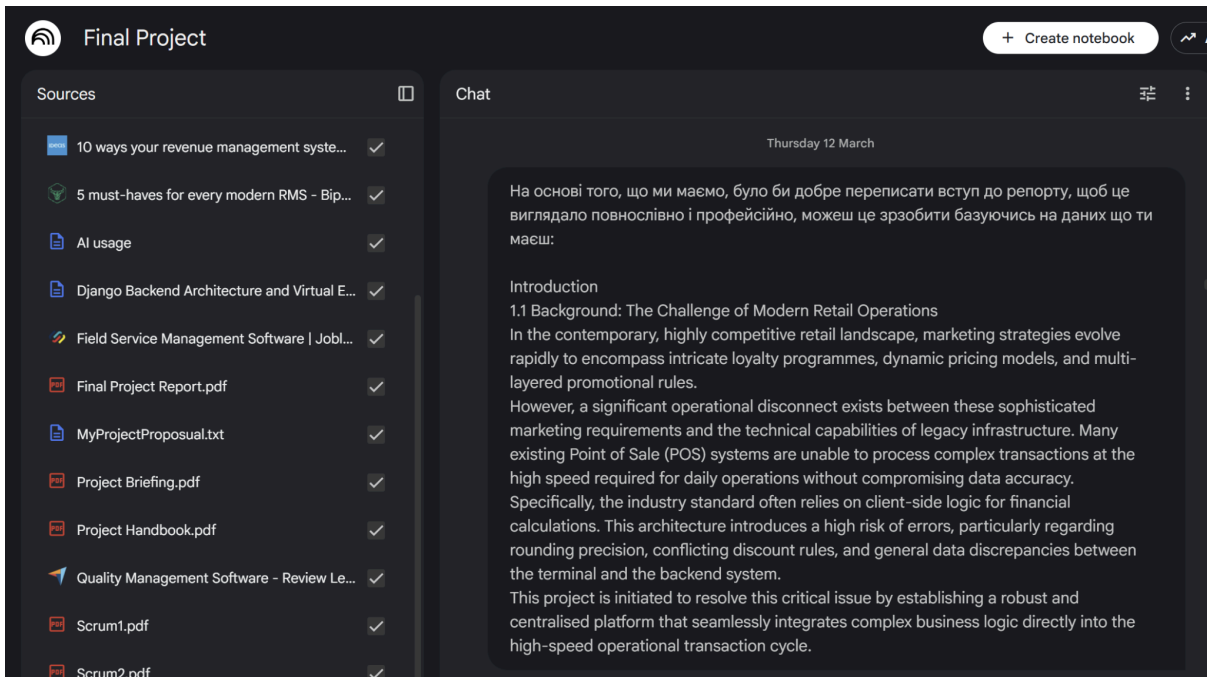
(ethical approval - Feedback Report)

Appendix F



Prompt	AI answer
Can you briefly describe what needs to be done for the interim report, given the requirements?	In general, you need to complete a Capstone Project that integrates the knowledge gained throughout the course. The project can be either workplace-based (if you work in IT) or an independent software solution. Here are the main stages and tasks to complete: 1. Project Lifecycle Stages: Inception Phase (Semester 4): Develop a concept, hold two meetings with the coordinator, create a draft and final project proposal (1–2 pages), and complete an ethics questionnaire...(cutted off)
Can you write a skeleton, bullet points for an interim report based on the project requirements and what I want to do and have actually already done?	Your Interim Report is an important milestone, as it is effectively an extended draft of the final report. Since you are developing Wynsum RMS (a cloud-based POS platform based on Django and React) and already have a clear vision of the

	<p>architecture and functionality, your report should reflect the results of the RAMP (Research, Analysis, Modeling and Planning) phase.</p> <p>Here is the skeleton of the report, adapted to your project and program requirements:</p> <p>1. Introduction</p> <p>Background: Describe the problem - the gap between complex retail marketing and operational speed.</p> <p>Project Goal: Create Wynsum RMS as a solution for automating discounts, taxes and financial logic on the server side.</p> <p>Scope: What exactly is included in the project (Promotional Engine, WebSocket architecture).</p> <p>2. Research and Analysis</p> <p>Requirements Analysis: Detailed description of the features you have already defined: recursive discounts, bundles, tax processing.</p> <p>Technical Research: Justification of stack choice (Python/Django, React) and use of WebSockets for real-time...(cutted off)</p>
--	---



Prompt	AI answer
Based on what we have, it would be a good	Here is a professionally rewritten and

idea to rewrite the introduction to the report to make it look more complete and professional, you can do this based on the data you have:

Introduction

1.1 Background: The Challenge of Modern Retail Operations

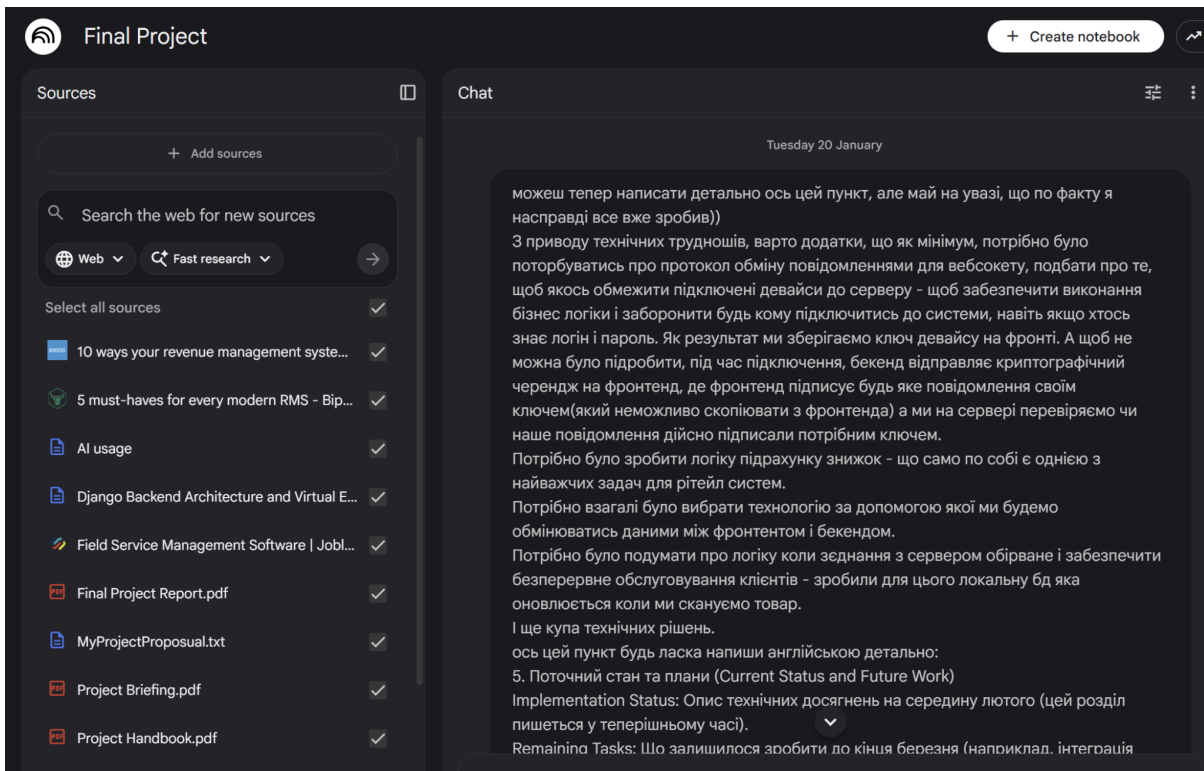
In the contemporary, highly competitive retail landscape, marketing strategies evolve rapidly to encompass intricate loyalty programmes...(cutted off)

expanded Background section for your interim report. This text combines your technical expertise with industry standards and the requirements of the Higher Diploma program, using the present tense to reflect the current state of the research.

1. Introduction

1.1 Background: The Challenge of Modern Retail Operations

In the contemporary, highly competitive retail and hospitality landscape, marketing strategies evolve rapidly to encompass intricate loyalty programs, dynamic pricing models, and multi-layered promotional rules.(cutted off)



Prompt	AI answer
<p>Can you now rewrite this point in detail, but keep in mind that in fact I have already done everything, so use the past tense?</p>	<p>Based on the project guidelines for the Interim Report and the technical details of your development, here is a detailed draft of Section 5. This section is written in the required present and future tenses to reflect the project's status as of mid-February.</p>
<p>Regarding technical difficulties, it is worth adding that at a minimum, it was necessary to worry about the messaging protocol for</p>	<p>5. Current Status and Future Work</p>

the web socket, to take care of somehow limiting the devices connected to the server - to ensure the execution of business logic and prohibit anyone from connecting to the system, even if someone knows the login and password. As a result, we store the device key on the front. And so that it cannot be forged, during connection, the backend sends a cryptographic challenge to the frontend, where the frontend signs any message with its key (which cannot be copied from the frontend) and we on the server check whether our message was really signed with the required key.(cutted off)

5.1 Implementation Status

As of February, Wynsum RMS has reached a significant level of functional maturity. The core architecture is established, following the plan developed during the RAMP phase . The current implementation status is as follows:

Backend Infrastructure: The Django server is fully operational, utilizing a modular structure with dedicated applications for the backoffice_api and pos_api

Database Integration: The relational schema is implemented in SQLite, supporting complex entities such as recursive discounts, multi-layered taxes, and item matrices

Promotional Engine Core: The server-side logic for calculating recursive discounts and product bundles is functional. This ensures that all financial calculations are performed centrally to maintain data integrity.(cutted off)

The screenshot displays a dark-themed application interface. At the top left, it says "Final Project" with a logo. On the top right, there is a "Create notebook" button. The interface is split into two main sections: "Sources" on the left and "Chat" on the right.

Sources Panel: It features a search bar "Search the web for new sources" with filters for "Web" and "Fast research". Below the search bar is a list of sources, each with a checkmark to its right, indicating they are selected. The sources include:

- 10 ways your revenue management syste...
- 5 must-haves for every modern RMS - Bip...
- AI usage
- Django Backend Architecture and Virtual E...
- Field Service Management Software | Jobl...
- Final Project Report.pdf
- MyProjectProposal.txt
- Project Briefing.pdf
- Project Handbook.pdf
- Quality Management Software - Review Le...
- Scrum1.pdf

Chat Panel: It shows a chat message in Ukrainian: "виходячи з цього, давай тепер перепишемо цю частину:". Below the message is a structured text block with the following sections:

- 1.2 Project Goal**
The core objective of this project is the development of Wynsum RMS (Retail Management System), a modern, cloud-connected RMS and POS platform engineered to automate and govern complex financial operations. To achieve maximum data integrity and reliability, the system is fundamentally designed to migrate all critical financial logic—including sequential or recursive discounts, product bundles, and all associated tax calculations—to the secure server side. This server-side architecture serves as the foundation for establishing a reliable "single source of truth" for every retail transaction, effectively eliminating the common errors and inconsistencies historically associated with traditional client-side calculation methods.
- 1.3 Scope and Technical Architecture**
The project defines the complete development of a modern, cloud-connected full-stack Retail Management System. The architecture is strategically chosen to ensure maximum reliability and high-speed operational performance. The system is built on a robust, three-tiered technical stack designed for scalability:
Backend (Server-Side Logic): Python/Django. Chosen for its "batteries-included" nature and efficiency in managing complex relational data models, Django provides a powerful, secure framework for all critical financial and operational logic.
Frontend (User Experience): React. The user interface is being developed with React to deliver a fast, responsive, and component-based experience for operators at the Point of Sale (POS) terminal.

At the bottom of the chat panel, there is a text input field "Start typing..." and a button labeled "15 sources" with a right-pointing arrow.

Prompt	AI answer
<p>based on the requirements, let's now rewrite this part:</p> <p>1.2 Project Goal The core objective of this project is the development of Wynsum RMS (Retail Management System), a modern, cloud-connected RMS and POS platform engineered to automate and govern complex financial operations. To achieve maximum data integrity and reliability, the system is fundamentally designed to migrate all critical financial logic—including sequential or recursive discounts, product bundles, and all associated tax calculations—to the secure server side. This server-side architecture serves as the foundation for establishing a reliable "single source of truth" for every retail transaction, effectively eliminating the common errors and inconsistencies historically associated with traditional client-side calculation methods.</p> <p>1.3 Scope and Technical Architecture The project defines the complete development of a modern, cloud-connected full-stack Retail Management System. The architecture is strategically chosen to ensure maximum reliability and high-speed operational performance.(cutted off)</p>	<p>Here is a professionally reworked and expanded version of sections 1.2 and 1.3 for your Interim Report. The text has been adapted to reflect the technical complexity you implemented within the Scrums, including asynchrony, Zero Trust security, and high-performance rendering.</p> <p>1.2 Project Goal The core objective of this project is the development of Wynsum RMS, a modern, cloud-connected Retail Management System engineered to serve concurrently as an authoritative "system-of-record" and an intelligent "productivity workspace". The platform is designed to automate and govern complex financial operations that legacy rule-based systems often fail to manage efficiently. To achieve maximum data integrity, the system follows a "Zero Trust" security architecture regarding the frontend. This necessitates migrating all critical financial logic—including sequential/recursive discounts, dynamic product bundles, and multi-layered tax calculations—exclusively to the secure server side . By establishing this server-side "single source of truth," the project eliminates the mathematical rounding errors, conflicting rules, and price manipulation risks historically associated with client-side calculation methods(cutted off)</p>
<p>can you also rewrite this part?, but since the focus is on the POS implementation, I have added two scrum files for you where you can see in detail what was done:</p> <p>4.2 Sprint Plan and Phasing The entire project development lifecycle is scheduled across seven distinct, two-week sprints, spanning from the initial RAMP phase in December to the final dissemination in April. This schedule is actively managed via Trello. Phase 1: RAMP and Design (Sprints 1–2): This initial phase has focused on foundational work, including full requirements analysis, technical research, and system design. Key deliverables have</p>	<p>Here is a revised and significantly expanded version of Section 4.2 Sprint Plan and Phasing, adapted specifically for the Final Report. The text is past tense (as required by the Final Report) and integrates the details of your specific sprints for the development of the POS terminal.</p> <p>4.2 Sprint Plan and Phasing The development lifecycle of the Wynsum POS Terminal was executed across seven distinct, two-week sprints. The schedule was actively managed via a Trello Kanban board, which served as the primary tool for tracking user stories, task priorities, and the "Definition of Done" for each iteration. Phase 1: RAMP and Architectural Design</p>

included the finalisation of the Entity-Relationship (ER) diagram and the cloud architecture.

Phase 2: Core Development (Sprints 3–5): The central development phase is dedicated to the implementation of the core technical stack. This involves building the Django backend, the Promotional Engine logic, and the React frontend components.

Phase 3: Refinement and Dissemination (Sprints 6–7): The final sprints are reserved for rigorous end-to-end testing, the full integration of the WebSocket architecture, and the preparation of dissemination materials.

(Sprints 1–2) The initial phase focused on established the foundational "Edge-Native" architecture. Key deliverables included the finalisation of the Entity-Relationship (ER) diagram and the design of the "Zero Trust" security model. During these sprints, research was conducted into ECDSA Device Verification to ensure that only registered hardware could interface with the Hub. The decision to decouple administrative tasks (backoffice_api) from terminal transactions (pos_api) was formalized here to ensure maximum operational speed.

Phase 2: Core POS Implementation...(cutted off)

Deep research chat links:

<https://gemini.google.com/share/cdb4bf138679>

<https://gemini.google.com/share/181e0512364a>

Appendices and References

Coyier, C. (2024) 'A Complete Guide to Flexbox', *CSS-Tricks*. Available at: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> (Accessed: 11 January 2026).

Django Software Foundation (2025) 'Serialization', *Django Documentation*. Available at: <https://docs.djangoproject.com/en/6.0/topics/serialization/> (Accessed: 05 January 2026).

Django Software Foundation (2025) 'Validators', *Django Documentation*. Available at: <https://docs.djangoproject.com/en/6.0/ref/validators/> (Accessed: 10 January 2026).

Forward, M. (2015) *Learning WebSockets: Design and Build Real-Time Web Applications*. 1st edn. Sebastopol, CA: O'Reilly Media.

Google NotebookLM (2026) AI response to prompts regarding report structuring and glossary. Available at: <https://notebooklm.google.com/> (Accessed: 31 March 2026).

Grammarly (2026) AI writing assistant for linguistic refinement. Available at: <https://www.grammarly.com/> (Accessed: 31 March 2026).

Google Gemini (2026) AI response to prompts regarding translation from Ukrainian to English and technical rephrasing. Available at: <https://gemini.google.com/> (Accessed: 31 March 2026).

MUI (2025) 'All components', *Material UI Documentation*. Available at: <https://mui.com/material-ui/all-components/> (Accessed: 11 January 2026).

React Hook Form (2025) 'Get Started', *React Hook Form Documentation*. Available at: <https://react-hook-form.com/get-started> (Accessed: 10 January 2026).

Remix Software (2025) 'Routing', *React Router Documentation*. Available at: <https://reactrouter.com/start/data/routing> (Accessed: 10 January 2026).