MARRYBROWN SALES AND PAYMENT ANALYTICS PLATFORM

YONG WERN JIE

UNIVERSITI TEKNOLOGI MALAYSIA

VALIDATION OF COLLABORATION

It is verified that this project is achieved through the collaboration between

MARRYBROWN SDN. BHD.

and

UNIVERSITI TEKNOLOGI MALAYSIA

Verified by (Industry): Date:

Signature:

Name: Dr. Noor Irliana Binti Mohd Rahim
Affiliation: MARRYBROWN SDN. BHD.
Official stamp:

Verified by (University): Date:

Signature:

Name: Dr. Noorfa Haszlinna binti Mustaffa
Affiliation: UNIVERSITI TEKNOLOGI MALAYSIA
Official stamp:

ABSTRACT

This project developed the Marrybrown Sales and Payment Analytics Platform to reduce operational risk associated with dependence on a third-party cloud point-of-sale (POS) reporting portal in a large-scale food and beverage (F&B) enterprise. In the original operating setting, Finance and Operations stakeholders relied on vendor-managed exports and portal interfaces for routine reconciliation and month-end closing, which constrained ad hoc analysis, limited transparency into underlying transactions, and could delay issue resolution when vendor services were degraded or unavailable. The implemented platform adopts a company-owned reporting redundancy architecture in which selected transactional tables are replicated 1:1 into a Microsoft SQL Server reporting environment, preserving raw data fidelity and traceability while supporting controlled refreshes, reruns, and late data adjustments. Report calculations were reconstructed in a semantic/API layer exposed through FastAPI, enabling schema-on-read transformations that keep business rules traceable and auditable while decoupling report delivery from direct database implementation details. The system was delivered through a React-based reporting portal that supports report searching, viewing, exporting, administrative controls, portal-triggered replication, data-quality checking, and ETL automation management. The project followed an iterative and incremental development approach in which report requirements were analysed from vendor portal behaviour and stakeholder feedback, implemented through Python-based ETL and API components, and validated through reverse engineering, reconciliation, and parity checking against vendor portal outputs for matched outlet, date range, and filter conditions. The project successfully delivered a company-controlled reporting platform for selected sales and payment reports, including additional customised reporting modules requested during implementation, while also documenting report areas that could not be fully closed because the remaining profit-related logic could not be isolated through the completed validation work. Overall, the platform improves internal report accessibility, strengthens control over reconstructed report logic, supports continuity during vendor reporting disruptions, and provides a maintainable basis for ongoing operational reporting.

ABSTRAK

Projek ini membangunkan Marrybrown Sales and Payment Analytics Platform bagi mengurangkan risiko operasi yang berkaitan dengan kebergantungan terhadap portal pelaporan sistem titik jualan (POS) berasaskan awan milik pihak ketiga dalam sebuah perusahaan makanan dan minuman (F&B) berskala besar. Dalam keadaan operasi asal, pihak Kewangan dan Operasi bergantung pada eksport serta antara muka portal yang diuruskan oleh vendor untuk proses pendamaian rutin dan penutupan hujung bulan, yang mengehadkan analisis ad hoc, mengurangkan ketelusan terhadap transaksi asas, dan boleh melambatkan penyelesaian isu apabila perkhidmatan vendor mengalami kemerosotan prestasi atau gangguan. Platform yang dilaksanakan menggunakan seni bina redundansi pelaporan milik syarikat, iaitu jadual transaksi terpilih direplikasi secara 1:1 ke dalam persekitaran pelaporan Microsoft SQL Server bagi mengekalkan kesetiaan data mentah dan kebolehkesanan, di samping menyokong proses penyegaran semula, larian semula, dan pelarasan terhadap data yang diterima lewat secara terkawal. Pengiraan laporan dibina semula dalam lapisan semantik/API yang didedahkan melalui FastAPI, sekali gus membolehkan transformasi schema-on-read yang memastikan peraturan perniagaan kekal boleh dikesan dan boleh diaudit, serta memisahkan penyampaian laporan daripada perincian langsung pelaksanaan pangkalan data. Sistem ini disampaikan melalui portal pelaporan berasaskan React yang menyokong carian, paparan, eksport, kawalan pentadbiran, replikasi yang dicetuskan melalui portal, semakan kualiti data, dan pengurusan automasi ETL. Projek ini mengikuti pendekatan pembangunan iteratif dan berperingkat, dengan keperluan laporan dianalisis daripada tingkah laku portal vendor dan maklum balas pihak berkepentingan, dilaksanakan melalui komponen ETL dan API berasaskan Python, serta divalidasi melalui kejuruteraan songsang, pendamaian, dan semakan kesepadanan terhadap output portal vendor bagi cawangan, julat tarikh, dan syarat penapisan yang sepadan. Projek ini berjaya menghasilkan platform pelaporan yang dikawal oleh syarikat untuk laporan jualan dan pembayaran terpilih, termasuk modul pelaporan tersuai tambahan yang diminta semasa pelaksanaan, di samping mendokumentasikan beberapa bahagian laporan yang tidak dapat ditutup sepenuhnya kerana baki logik berkaitan keuntungan tidak dapat diasingkan melalui proses validasi yang telah dijalankan. Secara keseluruhannya, platform ini meningkatkan kebolehcapaian laporan dalaman, mengukuhkan kawalan terhadap logik laporan yang dibina semula, menyokong kesinambungan semasa gangguan pelaporan vendor, dan menyediakan asas yang boleh diselenggara untuk pelaporan operasi yang berterusan.

TABLE OF CONTENT

	TITLE	PAGES
	ABSTRACT	**II**
	ABSTRAK	**III**
	TABLE OF CONTENT	**IV**
	LIST OF TABLES	**IX**
	LIST OF FIGURES	**XI**
Chapter 1	**INTRODUCTION**	**1**
1.1	Introduction	1
1.2	Problem Background	2
1.3	Problem Statement	3
1.4	Project Aim	3
1.5	Project Objectives	4
1.6	Project Scope	4
1.7	Project Importance	6
1.8	Report Organization	6
Chapter 2	**LITERATURE REVIEW**	**8**
2.1	Introduction	8
2.2	Fundamental Theory and Concepts	8
2.2.1	Sales and Payment Reporting in Vendor-Managed POS Environments	8
2.2.2	Data Warehousing and Replication-Based Analytical Stores	9
2.2.3	Data Integration Pipelines: ETL vs ELT and Idempotent Loads	10
2.2.4	Replication, Availability, and Consistency Considerations	10
2.2.5	Schema-on-Read, Database Design, Semantic Layers, and Service Interfaces	11

2.2.6 Reverse Engineering and Black-Box Validation of Legacy Reports 12

2.2.7 Data Quality and Reconciliation for Financial Reporting 12

2.2.8 Iterative Development Approaches for Evolving Requirements 13

2.3 Related Studies/Systems and Comparative Discussion 13

2.4 Workflow and Interface Considerations for Operational Reporting Portals 15

2.5 Technology Selection Considerations 17

2.6 Synthesis and Rationale Map 18

2.7 Summary 19

Chapter 3 METHODOLOGY 20

3.1 Introduction 20

3.2 Methodology Choice and Justification 20

3.3 Phases within the Iterative and Incremental Development Methodology 21

3.3.1 Phase 1: Requirement Analysis (Document Analysis and Stakeholder Feedback) 24

3.3.2 Phase 2: System Design 24

3.3.3 Phase 3: Implementation 25

3.3.4 Phase 4: Testing and Validation (Parity and Reconciliation) 26

3.3.5 Phase 5: Deployment 27

3.3.6 Phase 6: Review and Feedback 27

3.4 Project Schedule (Gantt Plan for the 40-Week Internship Period) 28

3.5 Implementation Environment Requirements 31

3.5.1 Hardware Requirements 32

3.5.2 Software Requirements 32

3.6 Summary 33

Chapter 4 ANALYSIS AND DESIGN 35

4.1 Introduction 35

4.2 System Analysis 35

4.2.1 Case Study Context (Continuity Reporting for Sales and Payments) 35
4.2.2 Stakeholders and Role-Based View 36
4.2.3 System Requirements Gathering Techniques 36
4.2.4 Use Case Diagram 37
4.2.5 User Requirement Outcomes 37
4.3 System Requirements 38
4.3.1 Functional Requirements (FR) 38
4.3.2 Non-Functional Requirements (NFR) 39
4.3.3 Constraints and Assumptions 40
4.4 Current System Analysis 40
4.5 System Design 41
4.5.1 System Architecture 41
4.5.2 Component Explanations 42
4.5.3 Report Coverage Overview 42
4.5.4 Data Engineering and API Design 45
4.5.4.1 Data Sources and Replication Boundary 45
4.5.4.2 Refresh Cadence and Idempotent Load Pattern 45
4.5.4.3 Operational Controls, Logging, and Data-Quality Checks 46
4.5.4.4 API Endpoint Pattern and Semantic Layer Responsibilities 47
4.5.4.5 Security Considerations 47
4.5.4.6 Report Reconstruction Overview 48
4.5.5 Database Design 48
4.5.6 Business Rule Reconstruction for the Formal Report Set 50
4.5.7 Interface Design 51
4.5.8 Summary 54

Chapter 5 IMPLEMENTATION AND TESTING 55
5.1 Introduction 55

5.2 System Development 55
5.2.1 Development Overview 55
5.2.2 Reporting Database and ETL Development 56
5.2.3 Backend API Development 57
5.2.4 Frontend Portal Development 58
5.2.5 Administrative and Operational Functions 60
5.2.6 Report Implementation Coverage 64
5.3 Coding of the system's main functions/Process 65
5.3.1 Data Replication and Refresh Process 65
5.3.2 Report Reconstruction Process 68
5.3.3 Selected Implementation Example: Payment Type (All Payment) 69
5.3.4 Selected Implementation Example: Voucher Campaign & Reward Sales Report 70
5.3.5 Query Performance Observation 71
5.3.6 Report Accuracy Validation 74
5.3.7 Black-Box Testing and User Acceptance Testing 82
5.3.8 White-Box Testing 91
5.4 Summary 100

Chapter 6 CONCLUSION 102
6.1 Introduction 102
6.2 System Contribution/Achievement 102
6.3 System Constraint 104
6.4 Future Suggestion 105
6.5 Summary 106

REFERENCES 108

Appendix A: Report Specifications 111
A.1 Formal Implementation and Business Scope Report Index 111
A.2 Additional Customised Reports 115

A.3 Detailed Report Specifications for the Formal 19-Report Scope 116

R01: Sale Delivery (By Sales Type) Ex Tax Calculation 116

R02: Payment Type (All Payment) 117

R03: Product Mix Report 117

R04: Delivery - FoodPanda, Grabfood, ShopeeFood 118

R05: Pickup & Declaration Report 119

R06: Stock Variance Report (Latest) 120

R07: Discount Remark Report 121

R08: Foodpanda Sales 122

R09: DELETED Items Report 122

R10: Sales Return Report 123

R11: [SOK] Each Kiosk Transaction Report 124

R12: Sales Cancelled Report 125

R13: Xilnex - Monthly Checking - COGS by Item (By Sales Type) 126

R14: Foodpanda Discount 126

R15: Mobile Ordering Sales 127

R16: Average SOS Report (New) 128

R17: MB Cash Voucher (with Barcode) Redemption Report 129

R18: MB Staff E - Voucher RM 20 & MB CASH VOUCHER RM10 (with Barcode) Redemption Report 130

R19: Product Mix with modifier without ETL 131

A.4 Detailed Specifications for Additional Customised Reports 131

C01: Roblox Free Chicken Burger Combo Sales 131

C02: Voucher Campaign & Reward Sales 132

C03: Promotion Item Additional Purchase Report 133

LIST OF TABLES

Table 1.1: High-level project scope boundary 5

Table 2.1: Comparative discussion of vendor-managed reporting versus company-owned reporting 14

Table 2.2: Reporting portal workflow and interface considerations for the implemented platform 17

Table 2.3: Synthesis / rationale map linking adopted choices to literature 19

Table 3.1: Phase-level inputs, activities, and evidence artefacts (per report module) 23

Table 3.2: Detailed Gantt-style project schedule across the 40-week internship period 31

Table 3.3: Hardware requirements (high-level) 32

Table 3.4: Software requirements (high-level) 33

Table 4.1: Stakeholder groups and reporting needs (role-based view) 36

Table 4.2: Functional requirements for the platform 39

Table 4.3: Non-functional requirements for the platform 39

Table 4.4: Component responsibilities and design considerations 42

Table 4.5: Formal implementation/business-scope report groups and design implications 44

Table 4.6: Additional customised reports and their relationship to the shared platform design 44

Table 4.7: Conceptual data dictionary (summary) 50

Table 5.1: Recorded timing observations for Payment Type (All Payment) 72

Table 5.2: Selected supporting timing observations across optimised report modules 73

Table A.1: Formal implementation and business scope report index 114

Table A.2: Additional customised reports implemented beyond the formal 19-report scope 115

Table A.3: Detailed specification for R01 116

Table A.4: Detailed specification for R02 117

Table A.5: Detailed specification for R03 118
Table A.6: Detailed specification for R04 119
Table A.7: Detailed specification for R05 120
Table A.8: Detailed specification for R06 121
Table A.9: Detailed specification for R07 121
Table A.10: Detailed specification for R08 122
Table A.11: Detailed specification for R09 123
Table A.12: Detailed specification for R10 124
Table A.13: Detailed specification for R11 125
Table A.14: Detailed specification for R12 125
Table A.15: Detailed specification for R13 126
Table A.16: Detailed specification for R14 127
Table A.17: Detailed specification for R15 128
Table A.18: Detailed specification for R16 129
Table A.19: Detailed specification for R17 129
Table A.20: Detailed specification for R18 130
Table A.21: Detailed specification for R19 131
Table A.22: Detailed specification for C01 132
Table A.23: Detailed specification for C02 133
Table A.24: Detailed specification for C03 134

LIST OF FIGURES

Figure 1.1: Comparison of Original Vendor-Dependent Reporting Arrangement and Implemented Internal Reporting Platform 3

Figure 3.1: Iterative Workflow for Analytics Platform Development 21

Figure 3.2: High-level system workflow (data replication to report delivery) 28

Figure 4.1: Use case diagram for the Marrybrown Sales and Payment Analytics Platform (illustrative) 37

Figure 4.2: Current vendor-dependent reporting workflow (simplified) 40

Figure 4.3: Logical architecture of the analytics platform 41

Figure 4.4: Boundary-based refresh and report-serving sequence (illustrative) 46

Figure 4.5: Simplified conceptual data model for sales and payment reporting (illustrative) 49

Figure 4.6: Report list interface (illustrative) 51

Figure 4.7: Reporting interface (illustrative) 52

Figure 4.8: Portal navigation and report workflow (illustrative) 52

Figure 4.9: Administrative interface overview (illustrative) 53

Figure 5.1: Reports Hub and representative report-query interface 59

Figure 5.2: User Management page 61

Figure 5.3: Data Sync page 62

Figure 5.4: Automation Control page 63

Figure 5.5: Portal-triggered reference-table refresh and execution tracking 66

Figure 5.6: Portal-triggered dated sales-data replication and data-quality review 67

Figure 5.7: Sanitised illustration of report reconstruction logic 69

Figure 5.8: PV01 Payment Type (All Payment) report accuracy validation 75

Figure 5.9: PV02 Sales Return Report accuracy validation 76

Figure 5.10: PV03 Sales Cancelled Report accuracy validation 77

Figure 5.11: PV04 Sale Delivery Ex Tax report accuracy validation 78

Figure 5.12: PV05 MB Cash Voucher redemption accuracy validation 79

Figure 5.13: PV06 Voucher Campaign and Reward Sales accuracy validation 80

Figure 5.14: PV07 Portal-triggered data quality validation 81

Figure 5.15: PV08 Product Mix and discount profit logic validation boundary 82

Figure 5.16: TC01 user login with valid credentials 83

Figure 5.17: TC02 Reports Hub access 84

Figure 5.18: TC03 report parameter loading 84

Figure 5.19: TC04 Payment Type report query 85

Figure 5.20: TC05 report totals review 85

Figure 5.21: TC06 report output export 86

Figure 5.22: TC07 advanced search behaviour 86

Figure 5.23: TC08 Sales Return Report query 87

Figure 5.24: TC09 customised report query 87

Figure 5.25: TC10 normal-user admin restriction 88

Figure 5.26: TC11 admin Data Sync access 88

Figure 5.27: TC12 manual sync request submission 89

Figure 5.28: TC13 sync progress review 89

Figure 5.29: TC14 data-quality re-check workflow 90

Figure 5.30: TC15 Automation Control review 90

Figure 5.31: TC16 User Management workflow 91

Figure 5.32: WB01 API parameter validation 92

Figure 5.33: WB02 Payment Type grouping logic 93

Figure 5.34: WB03 boundary-based ETL rerun 94

Figure 5.35: WB04 staged refresh safety 95

Figure 5.36: WB05 source-to-replica data quality check 96

Figure 5.37: WB06 quality repair request flow 97

Figure 5.38: WB07 report export mapping 98

Figure 5.39: WB08 role-based access restriction 99

Figure 5.40: WB09 automation-control request handling 100

Chapter 1

INTRODUCTION

1.1 Introduction

In contemporary retail operations, timely access to trusted transaction data is essential for operational oversight, financial control, and informed decision-making. In practice, many organisations consolidate operational data into analytical repositories such as data warehouses to support reporting, governance, and performance monitoring (Inmon, 2005; Kimball & Ross, 2013). Large-scale food and beverage (F&B) enterprises such as Marrybrown generate high volumes of sales and payment transactions across multiple outlets, which increases the importance of reliable report access, traceable report logic, and consistent data availability.

In the original operating arrangement, the organisation relied on a third-party cloud point-of-sale (POS) vendor for both transaction processing and analytical reporting. Sales and payment information was primarily accessed through vendor-managed portal views and exported files used by Finance and Operations stakeholders for routine review, reconciliation activities, and month-end closing. While this arrangement may reduce initial implementation effort, cloud computing guidance highlights that reliance on external service providers introduces availability, control, and security considerations that require active management (Badger et al., 2012; Jansen & Grance, 2011).

Accordingly, the organisational context of this project was defined by high transaction volume, dependence on vendor-managed reporting access, and the need for timely, trusted outputs for operational and financial workflows. This context established the need for a company-controlled reporting platform capable of reproducing required reports from replicated transactional data while improving continuity, traceability, and internal control.

1.2 Problem Background

The organisation's previous reporting arrangement depended heavily on a third-party POS service provider for report access and delivery. This created a practical single point of reporting dependency: when the vendor portal experienced downtime, degraded performance, or delayed report generation, Finance and Operations users could face disruption in obtaining required sales and payment information.

A further constraint was limited internal control over both the underlying datasets and the report logic applied to them. Access was mediated mainly through vendor interfaces and exported outputs, which restricted ad hoc investigation, limited transparency into report derivation, and reduced the organisation's ability to maintain or extend required reporting behaviour internally.

Internal reporting activities placed particular pressure on this arrangement during reconciliation and month-end closing periods, when timely access to trusted outputs was especially important. Delays or unavailability during these windows could interrupt verification workflows, slow issue resolution, and affect downstream financial preparation.

As illustrated in Figure 1.1, the original reporting arrangement provided a direct but vendor-dependent path from the external POS and reporting portal to Finance and Operations users. In contrast, the implemented Marrybrown Sales and Payment Analytics Platform introduced an internal reporting path through controlled data replication, a company-managed SQL Server reporting environment, a semantic/API layer, and a web-based reporting portal. This comparison highlights the shift from vendor-dependent report access to a company-controlled reporting architecture.

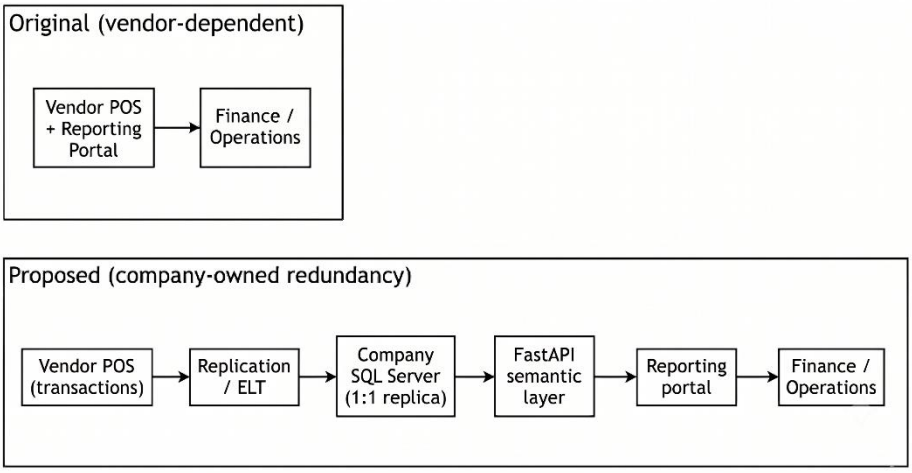

Figure 1.1: Comparison of Original Vendor-Dependent Reporting Arrangement and Implemented Internal Reporting Platform

1.3 Problem Statement

Based on the organisational context described above, this project addressed four interrelated problems. First, reliance on a third-party reporting portal limited organisational control over access to critical sales and payment reports. Second, restricted direct visibility into transactional data and report logic constrained ad hoc analysis, internal validation, and rapid issue investigation. Third, changes or extensions to required reports could not be maintained wholly within the organisation because report behaviour remained dependent on vendor-managed services. Finally, reporting disruption or delay during operationally sensitive periods, particularly reconciliation and month-end closing, could affect the timeliness and reliability of financial workflows.

1.4 Project Aim

The aim of this project was to develop the Marrybrown Sales and Payment Analytics Platform to provide company-controlled access to selected sales and

payment reports through replicated transactional data, reconstructed report logic, and an internal web portal.

1.5 Project Objectives

To achieve the stated aim, the following specific objectives have been defined:

i. To analyse stakeholder requirements for selected sales and payment reports, including continuity needs and report-level validation expectations.
ii. To design and implement a company-managed reporting environment in Microsoft SQL Server using 1:1 replication of required transactional data.
iii. To reconstruct and validate vendor-aligned report logic within a semantic/API layer so that selected reports can be reproduced with traceable business rules.
iv. To deliver a web-based reporting portal that supports report search, viewing, export, and administrative control for authorised internal users.

1.6 Project Scope

This project covers the end-to-end development of the Marrybrown Sales and Payment Analytics Platform, including data replication, report logic reconstruction, backend service development, portal delivery, and deployment support. The implementation scope focuses on sales and payment data required to reproduce selected operational and financial reports for internal use.

The platform uses a company-managed Microsoft SQL Server reporting environment to store replicated transactional data, a FastAPI backend to reconstruct

and deliver report logic, and a React-based portal to provide user access to validated reports. The implementation also includes controlled sync and ETL support, administrative functions for report operations, and validation activities to compare reconstructed outputs against the vendor portal under matched reporting conditions.

Where operational reporting needs evolved during implementation, the same platform architecture also supported additional company-requested reporting modules within the same sales and payment reporting domain.

The platform is intentionally limited to read-only reporting and does not perform write-back operations to the vendor POS environment. Broader enterprise analytics domains beyond the required sales and payment reporting surface, such as customer relationship management (CRM) and non-essential business intelligence extensions, are outside the project boundary. Detailed report specifications are presented in Appendix A.

Scope Area	**Coverage**
Data layer	Replication of selected sales and payment transaction data into a company-managed Microsoft SQL Server reporting environment
Report layer	Reconstruction and validation of selected vendor-aligned sales and payment reports through a FastAPI semantic/API layer
User layer	Internal web portal for report search, viewing, export, and administrative control
Boundary	Read-only reporting only; no write-back to the vendor POS system; broader analytics domains excluded from the current project

Table 1.1: High-level project scope boundary

1.7 Project Importance

This project is significant because it addressed an operational dependency in which access to critical sales and payment reports was mediated largely by vendor-managed services. By implementing a company-controlled reporting platform with replicated data and reconstructed report logic, the organisation gained greater continuity, transparency, and control over required reporting outputs. From an information quality perspective, the project emphasised reconciliation and parity validation to support the fitness-for-use of reproduced outputs for operational and financial reporting (Wang & Strong, 1996).

In addition, this work provides a structured case study on rebuilding vendor-dependent reports through replicated transactional data, a semantic/API delivery layer, and validation-driven implementation rather than treating reporting continuity as a simple interface duplication exercise. The resulting architecture also establishes a maintainable foundation for future internal reporting enhancement once the core sales and payment reporting requirements are stabilised.

The contribution of this project lies not only in delivering an internal reporting platform, but also in demonstrating a practical, validation-driven approach for reconstructing vendor-dependent operational reports using replicated transactional datasets in a real organisational environment.

1.8 Report Organization

Chapter 1 introduces the project context, problem statement, aim, objectives, scope, importance, and the organisation of the report. Chapter 2 reviews relevant literature on vendor dependency risks, replication and ETL approaches, schema-on-read and semantic layers, data quality and reconciliation, and iterative development approaches. Chapter 3 presents the methodology used to execute the project, including the validation approach and project workflow. Chapter 4 documents the analysis and design of the implemented system, including requirements, architecture, report

coverage overview, and design artefacts. Chapter 5 presents the implementation and testing of the system, including system development, main functions and processes, selected implementation examples, and validation activities. Chapter 6 concludes the report by summarising the system contribution, constraints, future suggestions, and overall project outcome.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

This chapter reviews scholarly and standards-based literature relevant to the Marrybrown Sales and Payment Analytics Platform as a company-owned reporting platform for an externally managed point-of-sale (POS) reporting environment. The review is structured to establish the theoretical and practical basis for the architecture and validation approach used in this project. Fundamental concepts are introduced first, followed by related studies and comparative discussion, workflow and interface considerations, and a brief discussion of the technology choices aligned to the literature.

2.2 Fundamental Theory and Concepts

2.2.1 Sales and Payment Reporting in Vendor-Managed POS Environments

In vendor-managed POS environments, transaction processing and report access may be delivered through the same externally managed service boundary. From a cloud-computing perspective, this arrangement places operational reporting within a third-party availability, security, and change-management context rather than wholly under organisational control (Mell & Grance, 2011; Badger et al., 2012; Jansen & Grance, 2011). For sales and payment reporting, this matters because daily totals, payment breakdowns, and reconciliation outputs are often consumed through vendor-managed portal interfaces and exports rather than through an internally operated reporting stack.

The literature also highlights lock-in effects in cloud adoption, where switching providers or rebuilding capabilities can incur high technical and operational costs (Armbrust et al., 2010). These costs are amplified when report logic exists primarily in a vendor-managed portal and is not fully documented or reproducible by the organisation. Consequently, continuity-oriented initiatives commonly prioritise retaining organisational access to usable transaction data and establishing reporting capabilities that are less exposed to vendor-side service disruptions (Badger et al., 2012; Armbrust et al., 2010).

2.2.2 Data Warehousing and Replication-Based Analytical Stores

Data warehousing literature frames a data warehouse as an integrated repository designed to support reporting and analysis rather than operational transaction processing (Inmon, 2005; Kimball & Ross, 2013). For sales and payment analytics platforms, this repository forms the database foundation for storing duplicated transactional data in a form that remains queryable for reporting while preserving a clear relationship to operational records. Traditional designs frequently apply modelling and transformation choices (e.g., dimensional modelling) to optimise analytical queries, but such transformations can increase complexity and may obscure traceability to source records if governance is weak.

For reporting continuity, an alternative design is a fidelity-first replicated analytical store that preserves the structure and content of source transactional tables. This approach prioritises traceability and simplifies reconciliation by reducing ambiguity about how report outputs relate to replicated source records. In practice, the choice between an optimised dimensional model and a replicated fidelity-first store depends on the primary objective: performance optimisation versus assured traceability and parity with externally generated reports (Inmon, 2005; Kimball & Ross, 2013).

2.2.3 Data Integration Pipelines: ETL vs ELT and Idempotent Loads

Data integration pipelines are commonly described using ETL (extract-transform-load), where data is transformed prior to loading into the target repository (Kimball & Caserta, 2004). In sales and payment analytics platforms, these pipelines form the process layer that moves vendor-originated data into the organisation-controlled reporting environment. Contemporary data platform practices also adopt ELT patterns, in which raw data is loaded first and transformations are applied in the target environment. This aligns with schema-on-read approaches that preserve raw data while enabling flexible downstream transformations (Azzabi et al., 2024).

For operational reliability, pipeline design must consider rerun safety and operational recovery. Idempotent loading patterns reduce the risk of duplicate records and support repeatable refresh cycles when failures occur or when late data corrections are detected. In practice, idempotency is achieved through controlled load boundaries (e.g., date-range refresh) and deterministic processing rules that yield the same replicated state for the same boundary.

2.2.4 Replication, Availability, and Consistency Considerations

Replication is used to improve availability and support resilience by maintaining copies of data across systems. In distributed and replicated systems, trade-offs between consistency, availability, and operational complexity are central design considerations (Kleppmann, 2017). For reporting continuity platforms, strict real-time consistency is typically not required; instead, the practical objective is timely access to sufficiently recent and internally consistent data, supported by a refresh process and validation evidence that stakeholders can trust during reporting-intensive periods.

Accordingly, replication design in reporting contexts is commonly paired with operational controls (e.g., rerun capability, logging, and post-load checks) to manage failure recovery and to provide confidence in the replicated state. These controls

support both day-to-day reporting and the investigative workflows that arise when discrepancies are detected.

2.2.5 Schema-on-Read, Database Design, Semantic Layers, and Service Interfaces

Schema-on-read practices emphasise preserving raw or lightly transformed datasets and applying transformations at consumption time, which supports iterative refinement of business logic while maintaining traceability (Azzabi et al., 2024). For reporting platforms that duplicate operational sales and payment data, this principle supports database designs in which replicated tables remain close to source structure, while report-specific calculations are handled separately for auditability and change control (Inmon, 2005; Kimball & Ross, 2013).

Data repository design and semantic design are therefore related but distinct concerns. Data warehousing literature differentiates between storage structures used to retain and organise data and presentation or consumption structures used to deliver consistent analytical views (Inmon, 2005; Kimball & Ross, 2013). In this project context, reconstructed report logic is treated as semantic-layer logic: duplicated transactional data are retained for traceability, while business rules, derived fields, and report-specific aggregation behaviour are centralised above the database layer to reduce metric drift and simplify maintenance.

For service delivery, REST is a widely cited architectural style for network-based systems, emphasising stateless communication, uniform interfaces, and resource-oriented design (Fielding, 2000). A RESTful API therefore provides a standard mechanism to deliver report outputs to a portal while decoupling report consumption from underlying database implementation details and keeping business-rule changes versioned and auditable at the service layer.

2.2.6 Reverse Engineering and Black-Box Validation of Legacy Reports

When vendor report logic is proprietary and not fully documented, reverse engineering and design recovery provide a structured lens for inferring behaviour from observable artefacts (Chikofsky & Cross, 1990). In such settings, black-box testing methods support parity validation by comparing system outputs against vendor exports under identical parameters (Myers et al., 2011). For reporting platforms, this process extends beyond matching totals; it also requires validating calculation rules and edge-case handling (e.g., returns, voids, split tenders, and rounding behaviour) to reduce operational reconciliation risk.

2.2.7 Data Quality and Reconciliation for Financial Reporting

Data quality is commonly framed as fitness for use, where suitability depends on the consumer's needs and the use context (Wang & Strong, 1996). Standards and reference models describe core quality characteristics such as accuracy, completeness, consistency, and timeliness (International Organization for Standardization, 2008). In sales and payment reporting, these characteristics are directly relevant because stakeholders require totals, breakdowns, and reconciliation outputs that are dependable for decision-making and audit preparation.

Data quality management literature discusses practical techniques for assessing and improving data quality, including profiling and validation rules (Batini & Scannapieco, 2006). In replicated reporting contexts, these techniques are operationalised through reconciliation checks (e.g., cross-report consistency of totals) and repeatable parity comparisons against vendor exports for agreed validation windows.

2.2.8 Iterative Development Approaches for Evolving Requirements

Iterative development approaches emphasise incremental delivery and feedback-driven refinement (Beck et al., 2001). Scrum formalises iterative planning and review through time-boxed cycles and continuous refinement (Schwaber & Sutherland, 2020). In software engineering practice, iterative approaches are commonly recommended when requirements are uncertain or when continuous validation is required to manage risk (Pressman & Maxim, 2014; Sommerville, 2015). In report reconstruction projects, requirements and edge cases often emerge during parity validation, which makes iterative delivery suitable for progressively stabilising business rules while retaining validation evidence.

2.3 Related Studies/Systems and Comparative Discussion

Related literature on vendor-managed reporting environments, organisation-controlled reporting repositories, and schema-on-read analytical platforms provides a comparative basis for the architecture adopted in this project. Cloud computing literature recognises vendor dependency and lock-in as concerns that can affect cost, portability, governance, and operational control (Armbrust et al., 2010; Badger et al., 2012). Data warehousing literature explains how organisation-controlled repositories can improve reporting reliability and analytical access (Inmon, 2005; Kimball & Ross, 2013). Survey work on data lakes and schema-on-read practices further highlights the value of preserving raw data and applying transformations flexibly at the consumption layer (Azzabi et al., 2024).

However, fewer sources provide detailed guidance on reconstructing proprietary vendor reporting logic while demonstrating report-level parity through defensible validation artefacts. This creates a practical gap for organisations that require continuity reporting but must operate without full access to vendor calculation definitions.

Table 2.1 provides a comparative discussion between a vendor-managed reporting portal and a company-owned reporting platform, highlighting the trade-offs that motivate the architecture adopted in this project.

Criterion	**Vendor-managed portal only**	**Company-owned reporting platform**
Availability and continuity	Reporting access depends on vendor service availability and change management (Badger et al., 2012; Jansen & Grance, 2011).	Provides an alternative reporting path by operating on replicated data under organisational control (Kleppmann, 2017).
Data transparency and traceability	Users typically interact through portal views and exports; traceability to raw transactions may be limited by vendor interface constraints.	Fidelity-first replication supports traceability from report outputs to replicated source records (Inmon, 2005).
Report logic control	Report definitions and changes are vendor-controlled; organisational adaptation is constrained.	Business rules are reconstructed and versioned in a semantic layer, enabling auditable updates.
Validation burden	Vendor portal is treated as the reference, but internal validation is limited to exported outputs.	Requires black-box parity validation and reconciliation evidence to build trust in reconstructed logic (Myers et al., 2011; Wang & Strong, 1996).
Cost and complexity	Lower internal build complexity, but potential lock-in and dependency costs (Armbrust et al., 2010).	Higher engineering and validation effort initially; benefits depend on sustained operational use and governance.

Table 2.1: Comparative discussion of vendor-managed reporting versus company-owned reporting

Data warehousing literature, together with research on schema-on-read practices, also motivates a design trade-off between transformation-heavy analytical models and replication-first designs. Dimensional modelling supports analytical performance and standardised business views, whereas replication-first designs prioritise traceability and parity alignment to source representations (Inmon, 2005; Kimball & Ross, 2013). For continuity-oriented reporting where parity and reconciliation are primary goals, the architecture used in this project adopts replication for fidelity and applies transformations at the semantic layer, consistent with schema-on-read principles (Azzabi et al., 2024).

Based on this synthesis, the project architecture combines a fidelity-first replicated datastore to support traceability and reconciliation, an ELT workflow with idempotent refresh to support safe reruns and late adjustments, a semantic/API layer for centralised business rules and service delivery, reverse engineering with black-box parity validation to infer and validate proprietary behaviour, and iterative development practices to manage emerging edge cases (Azzabi et al., 2024; Beck et al., 2001; Chikofsky & Cross, 1990; Kleppmann, 2017; Myers et al., 2011; Wang & Strong, 1996).

2.4 Workflow and Interface Considerations for Operational Reporting Portals

In operational sales and payment reporting, interaction is typically organised around a report-retrieval workflow rather than around dashboard-style monitoring. Enterprise reporting systems commonly present users with a catalogue of available reports, after which the user selects a report, specifies the required parameters, executes the query, reviews the returned output, and exports the result if needed. Oracle MICROS Reporting and Analytics reflects this pattern through report-list navigation, prompt-based parameter entry, filtering, drill-down behaviour, and export functions for generated reports (Oracle, 2023). A comparable workflow is also evident in paginated reporting systems, where users select a report, enter parameters before rendering the output, and export the result in formats such as Excel or PDF (Microsoft,

2026). For this type of use, the display purpose is primarily detailed lookup and operational reporting, which makes tabular presentation especially appropriate when users need precise values for verification, reconciliation, and follow-up action (Few, 2012).

Usability literature on self-service portals further suggests that routine operational tasks benefit from clear navigation, understandable prompts, consistent interaction patterns, and reduced effort in repeated use (Matloobtalab & Ferati, 2025). In addition, Shneiderman's information-seeking framework highlights the practical importance of filtering and details-on-demand when users must narrow a large report space or inspect a result set progressively according to immediate task needs (Shneiderman, 1996). Within a reporting portal, these considerations translate into structured report selection, predictable parameter controls, and result views that support progressive inspection without unnecessary visual or procedural complexity.

Taken together, these sources suggest that a sales and payment reporting portal should prioritise report discoverability, parameter-driven retrieval, readable tabular outputs, and export support for reconciliation-oriented work. This interpretation is consistent with the practical workflow observed in vendor-managed reporting environments and provides a conceptual basis for the portal requirements and interface decisions discussed in Chapter 4. Table 2.2 summarises the reporting portal design considerations most relevant to the implemented platform.

Design consideration	Supporting source(s)	Relevance to the implemented platform
Report catalogue and navigation	Oracle (2023); Matloobtalab & Ferati (2025)	Users should be able to locate relevant reports efficiently from a structured list or menu.
Parameter specification before query execution	Oracle (2023)	Report retrieval should support explicit parameter entry such as outlet, date range, and status before output generation.

Readable tabular output for operational checking	Few (2012)	Sales and payment reporting often requires exact values for checking, reconciliation, and follow-up action rather than visual summaries alone.
Filtering and progressive inspection of results	Shneiderman (1996); Oracle (2023)	Users should be able to narrow report outputs and inspect relevant details without unnecessary interface complexity.
Consistency of prompts, labels, and interaction flow	Matloobtalab & Ferati (2025)	Consistent controls across reports reduce learning effort and improve usability for recurring operational tasks.
Export support for reconciliation and offline review	Oracle (2023); Microsoft (2026)	Retrieved report outputs should be exportable to support reconciliation, sharing, and further offline review.

Table 2.2: Reporting portal workflow and interface considerations for the implemented platform

2.5 Technology Selection Considerations

Within the implemented platform, Microsoft SQL Server, a Python-based ELT workflow, FastAPI, and React were selected as engineering technologies that fit the literature-grounded architectural direction established in this chapter. SQL Server supports the company-managed reporting repository, the Python-based ELT workflow supports controlled extraction and refresh, FastAPI supports RESTful service delivery for reconstructed reports, and React supports parameter-driven retrieval, tabular viewing, and export workflows for internal users (FastAPI, n.d.; React, n.d.; Fielding, 2000). This section is included to connect the literature-grounded architecture to the actual technology stack adopted in the project, while detailed implementation discussion is reserved for Chapters 4 and 5.

2.6 Synthesis and Rationale Map

Table 2.3 synthesises how the literature reviewed in this chapter supports the architectural and methodological choices adopted in this project. The synthesis links each major design or methodological choice to its underlying academic rationale, thereby making the basis for the implemented platform explicit and traceable.

Method/design choice	**Rationale grounded in literature**	**Key sources**
Iterative development for report reconstruction	Requirements and edge cases emerge during parity validation; iterative delivery supports risk reduction through frequent feedback and refinement.	Beck et al. (2001); Schwaber & Sutherland (2020); Pressman & Maxim (2014); Sommerville (2015)
ELT workflow with idempotent refresh	Loading raw data first supports schema-on-read transformations and iterative rule refinement; idempotent boundaries support reruns and recovery.	Kimball & Caserta (2004); Azzabi et al. (2024)
Fidelity-first replication for traceability	Preserving source structure supports reconciliation and explainability of outputs, which is central in financial reporting contexts.	Inmon (2005); Kleppmann (2017)
Consistency/availability framing for reporting	Reporting prioritises timely, consistent snapshots rather than strict real-time consistency; operational controls improve reliability.	Kleppmann (2017)
Schema-on-read with semantic layer exposed via API	Separating duplicated storage structures from business-rule and service-interface logic	Inmon (2005); Kimball & Ross (2013); Azzabi et al.

	preserves traceability, reduces metric drift, and supports auditable evolution.	(2024); Fielding (2000)
Reverse engineering + black-box parity validation	Proprietary vendor logic requires inference from observable behaviour; black-box testing provides empirical parity evidence.	Chikofsky & Cross (1990); Myers et al. (2011)
Data quality and reconciliation checks	Fitness-for-use for financial reporting depends on accuracy, completeness, and consistency; reconciliation operationalises quality assurance.	Wang & Strong (1996); International Organization for Standardization (2008); Batini & Scannapieco (2006)
Cloud dependency risk and lock-in considerations	Vendor dependence can introduce availability/security/change-management risks and lock-in costs; continuity designs mitigate exposure.	Mell & Grance (2011); Badger et al. (2012); Jansen & Grance (2011); Armbrust et al. (2010)

Table 2.3: Synthesis / rationale map linking adopted choices to literature

2.7 Summary

This chapter reviewed key concepts and prior work relevant to vendor-managed sales and payment reporting environments, replication-based analytical stores, ELT workflows, schema-on-read and database design, semantic layers, workflow and interface considerations for operational reporting portals, reverse engineering and black-box validation, data quality and reconciliation, and iterative development. Taken together, the literature supports an internal reporting platform that prioritises data fidelity, reconciliation, controlled report delivery, and iterative refinement. The next chapter presents the methodology used in this project to implement and validate that approach.

Chapter 3

METHODOLOGY

3.1 Introduction

This chapter describes the methodology used to develop the Marrybrown Sales and Payment Analytics Platform. The project combined reverse engineering of vendor-managed POS reports, a 1:1 replication and ELT workflow into a company-managed reporting database, and report-level parity validation through iterative comparison and reconciliation. Because the underlying vendor report logic was not fully documented, the methodology prioritised incremental delivery, repeatable validation cycles, staged deployment, and retention of evidence artefacts that could support traceability across analysis, design, implementation, and testing activities.

3.2 Methodology Choice and Justification

The project adopted an iterative and incremental development approach aligned with Agile principles, where functionality is delivered in small increments and continuously validated against expected outputs (Beck et al., 2001; Schwaber & Sutherland, 2020). This choice is suitable for report reconstruction because requirements, calculation edge cases, data-boundary issues, and workflow refinements emerge progressively during parity validation, and the cost of rework is reduced when changes are applied in smaller validated iterations (Pressman & Maxim, 2014; Sommerville, 2015). The same approach also supported gradual stabilisation across the replication layer, semantic/API layer, and portal layer, rather than treating the system as a one-pass build. The academic rationale for iterative delivery, ELT, replication-first traceability, and reconciliation-driven validation is synthesised in Chapter 2 (see Table 2.3).

3.3 Phases within the Iterative and Incremental Development Methodology

Work was organised as an iterative and incremental cycle that repeated for each targeted report module and supporting platform function. As illustrated in Figure 3.1, each cycle moved through six recurring phases: requirement analysis, system design, implementation, testing and validation, deployment, and review and feedback, before returning to requirement analysis when additional refinement was required. The phases were structured to preserve traceability from requirement interpretation to implementation and validation evidence, while allowing supporting operational functions such as controlled sync, quality checking, and scheduled automation to be introduced after core report logic had been sufficiently stabilised. Table 3.1 summarises the typical inputs, activities, and evidence artefacts produced in each phase.

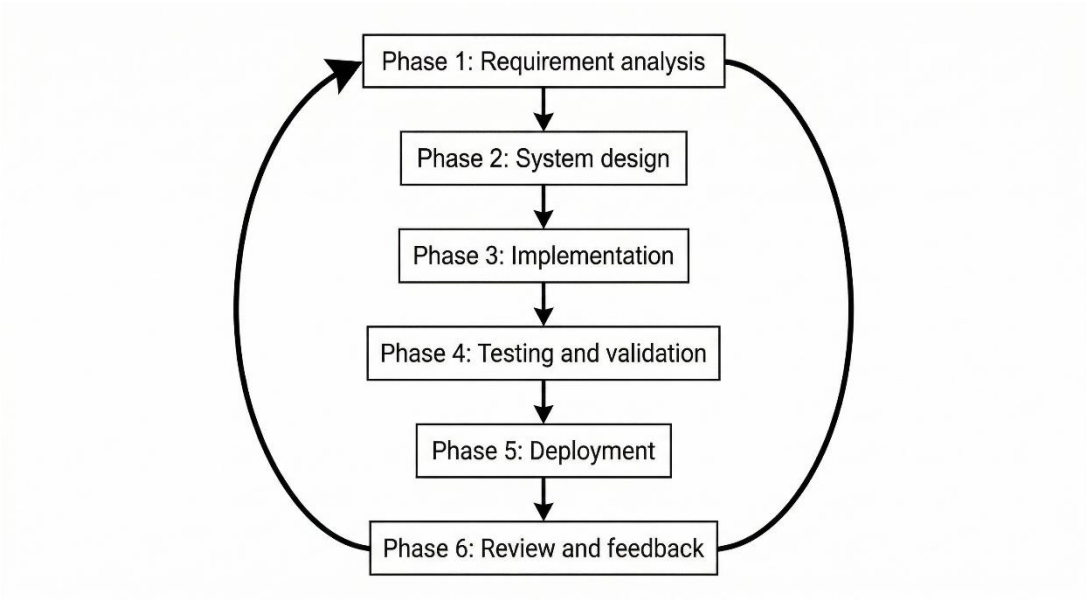

Figure 3.1: Iterative Workflow for Analytics Platform Development

Phase	Typical inputs	Key activities	Outputs / evidence artefacts (examples)

Phase 1: Requirement analysis	Vendor portal report views, exported files, parameter settings, stakeholder clarifications	Identify parameters and expected breakdowns; define output fields and totals; catalogue edge cases; draft data-to-report mapping; define acceptance criteria	Parameter matrix; output-field inventory; baseline export pack; acceptance checklist
Phase 2: System design	Phase 1 artefacts, replicated schema, constraints and non-functional requirements	Specify replication scope and join paths; define refresh boundary and rerun handling; define API contract and export behaviour; define validation plan and test cases	Report specification draft; API contract; data mapping; test plan; design diagram references
Phase 3: Implementation	Design artefacts, replicated datasets, acceptance criteria	Implement version-controlled report queries and rules; build API endpoint; implement parameter validation and formatting; add logging and error handling	Endpoint implementation; reproducible queries; sample outputs; execution logs

Phase 4: Testing and validation	Baseline exports, platform outputs, acceptance checklist	Compare outputs; investigate discrepancies; refine rules; run regression checks; record acceptance decisions and reconciliation results	Comparison evidence; discrepancy log; updated rule versions; validation records
Phase 5: Deployment	Validated module, environment-ready checklist	Deploy validated changes; configure access controls; update release documentation; prepare controlled operational use	Deployed endpoint or portal view; deployment checklist; release notes
Phase 6: Review and feedback	Deployed module, stakeholder feedback	Capture usability gaps and missing cases; triage defects versus enhancements; update acceptance criteria and test scenarios	Feedback log; change requests; revised acceptance criteria

Table 3.1: Phase-level inputs, activities, and evidence artefacts (per report module)

The phase descriptions in Sections 3.3.1 to 3.3.6 elaborate the responsibilities summarised in Table 3.1 and explain how each phase contributed to parity-driven report reconstruction and evidence retention.

3.3.1 Phase 1: Requirement Analysis (Document Analysis and Stakeholder Feedback)

Requirements were elicited through document analysis of vendor portal reports and exported files, together with stakeholder clarification from Finance and Operations users regarding business intent, expected breakdowns, and acceptance expectations for continuity reporting. Because vendor logic was treated as a black box, the project applied a reverse engineering perspective to infer business rules from observable inputs and outputs (Chikofsky & Cross, 1990). Black-box testing principles supported this phase by enforcing comparable parameters such as outlet and date range for subsequent parity checks (Myers et al., 2011). In practical terms, this phase produced a parameter matrix, an output-field inventory, and a baseline export pack that served as the reference set for later comparisons. Edge cases that could materially affect parity, such as voids, returns, split tenders, and rounding behaviour, were recorded as test scenarios to support repeatable validation as rules evolved.

Key outputs of Phase 1 therefore included a report-specific parameter list, an output-field inventory, a draft data-to-report mapping, and an acceptance checklist describing what constituted an acceptable parity match for the report. These artefacts provided the basis for both implementation planning and validation evidence retention in later phases.

3.3.2 Phase 2: System Design

System design translated Phase 1 requirements into design artefacts across the platform layers. At the replication and data-preparation level, this included determining the minimum replication boundary for each targeted report by tracing required output columns and business rules back to the relevant source tables, join paths, and reference data. The replicated schema preserves a 1:1 table structure with the source database, but only the tables required for the targeted reports are selected for replication. In the project context, transactional data replication is generally bounded from 2025 onward in line with organisational storage constraints, whereas

reference tables are replicated in full. The design also defined refresh boundaries, rerun considerations, and the operational approach for controlled portal-triggered sync, source-to-replica quality checks, and scheduled ETL functions.

At the semantic/API and portal layers, the design specified both the report contract and the report presentation in a traceable manner. This included endpoint naming, required and optional parameters, response schema, output columns, totals and subtotals, sorting and formatting rules, and export behaviour aligned to vendor report structures. Report design also defined the use of reusable shared components and a common layout across the portal, while allowing report-specific configuration for parameters, column sets, and output behaviour. The design further established a validation plan by mapping acceptance criteria and edge-case scenarios to concrete test cases and evidence artefacts such as baseline exports and comparison outputs. Design modelling artefacts such as architecture diagrams, data flow, and simplified data model views are presented in Chapter 4, while detailed per-report specifications are consolidated in Appendix A.

3.3.3 Phase 3: Implementation

Implementation followed a logic-first sequence: report logic was developed at the semantic/API layer and exercised using controlled parameters aligned to Phase 1 baseline exports. Where feasible, query prototypes were first validated against the replicated schema to confirm data availability, join correctness, and boundary handling, after which logic was stabilised into version-controlled queries and exposed through an API endpoint with a defined request and response contract. Implementation work also included consistent parameter validation, deterministic output ordering where required for comparison, and export-ready formatting so that parity checks could be performed reliably against vendor exports. Logging and error handling were incorporated to support traceability of runs during iterative validation cycles.

During early iterations, controlled manual sync and rerun procedures were used to stabilise replicated data windows and report logic. After the replication flow and

validation approach had matured, the platform was extended with scheduled ETL capabilities, administrative automation controls, and data-quality checking support so that the validated reporting environment could support both portal-triggered sync operations and managed automation.

3.3.4 Phase 4: Testing and Validation (Parity and Reconciliation)

Testing and validation were performed at both module and system levels. At the module level, parity validation was conducted by generating platform outputs using identical parameters to vendor exports and comparing results at the appropriate granularity, including row-level comparison where applicable and aggregate-level comparison where exports grouped outputs. Comparisons were performed using consistent filtering and boundary rules, and discrepancies were investigated and resolved through iterative refinement of filters, joins, date handling, and edge-case treatment. To reduce regression risk, previously validated scenarios, including recorded edge cases, were rechecked after rule changes so that fixes for one report did not introduce unintended drift in another.

At the system level, reconciliation checks were applied to confirm cross-report consistency, such as alignment between sales totals and payment totals for the same reporting windows under defined rules. Source-to-replica quality checks were also used for defined sync windows to confirm that replicated data remained fit for report reconstruction and investigation work. Together, these checks operationalised data quality perspectives such as accuracy, completeness, and consistency (International Organization for Standardization, 2008; Wang & Strong, 1996). Validation artefacts, including baseline exports, comparison outputs, and acceptance decisions, were retained to support repeatability.

The validation methodology also included black-box comparison against vendor outputs, white-box review of reconstructed business rules and query behaviour, and user-oriented verification of report retrieval, viewing, and export functions. In addition, a basic operational performance evaluation was used to assess usability under

typical usage conditions. Response-time observations for representative report queries were considered in relation to practical expectations for routine Finance and Operations workflows. The intent of this evaluation was to confirm operational acceptability rather than to conduct exhaustive benchmarking or load testing.

3.3.5 Phase 5: Deployment

Upon meeting validation criteria for a module, the corresponding API and portal changes were deployed to the target environment according to an environment-ready checklist. Deployment activities included configuration validation such as database connectivity and access controls, release documentation to support traceability of rule changes, and rollback considerations for changes that materially affected report outputs. In this project, deployment was staged so that validated report modules and supporting platform functions could first be checked in a controlled environment and then consolidated into the broader implemented platform environment. This phase also covered the operationalisation of controlled sync procedures, scheduled ETL functions, and related administrative controls for report operations.

3.3.6 Phase 6: Review and Feedback

Feedback was collected from Finance and Operations users to identify usability issues, missing data elements, and any residual discrepancies observed during real workflows. Feedback was documented as traceable change requests and triaged into parity defects that required rule correction and enhancements that adjusted workflows or presentation without changing core financial meaning. Where feedback changed acceptance criteria or revealed new edge cases, the iteration returned to Phase 1 for the affected module so that requirements artefacts could be updated and the revised logic validated again with retained evidence.

As shown in Figure 3.2, the high-level operational workflow of the implemented platform began with data extraction and replication from the external POS environment, followed by ELT processing into the company-owned SQL Server reporting database, source-to-replica quality checking, semantic/API-based report reconstruction, and final report delivery through the internal reporting portal to Finance and Operations users. This view links the development phases described above to the end-to-end technical flow implemented and validated during the project.

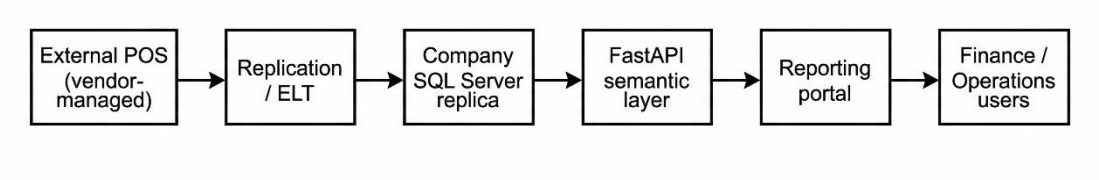

Figure 3.2: High-level system workflow (data replication to report delivery)

3.4 Project Schedule (Gantt Plan for the 40-Week Internship Period)

For planning and execution purposes, the overall project was organised across a 40-week internship period. Because the project followed an iterative and incremental methodology, the schedule was not treated as a strictly linear one-pass sequence in which all analysis ended before all design, and all design ended before all implementations. Instead, the work was arranged as overlapping blocks in which early planning activities established the foundation, while later report reconstruction, validation, deployment, and refinement cycles revisited earlier findings as new report behaviours and operational issues emerged. Table 3.2 therefore presents a detailed week-based Gantt-style plan that reflects both sequencing and overlap across the internship period.

Work package	**Weeks (planned)**	**Main focus**	**Relation to the iterative and incremental methodology**
Project initiation and company briefing	1-2	Understand business context, reporting	Establishes the initial problem boundary

		dependency, internship scope, and stakeholder expectations	before the first iteration cycle begins
Problem formulation and scope definition	2-4	Define the reporting problem, project boundary, and high-level objectives	Produces the baseline scope used for early iterations, while allowing later refinement
Literature review and source collection	3-8	Review references on vendor dependency, replication, ELT, validation, and reporting portals	Builds the academic basis that informs repeated design and validation decisions
Methodology definition and project planning	5-8	Define the iterative and incremental approach, report-validation method, and initial work structure	Formalises the cycle used repeatedly for report reconstruction modules
Current system study and report requirement analysis	6-12	Analyse vendor portal behaviour, report parameters, output structure, and acceptance expectations	Supplies the first iteration backlog and is revisited when new edge cases appear
System analysis and high-level design	9-14	Prepare architecture, data-flow, and interface design foundations	Creates the initial technical blueprint for implementation iterations
Replication scope definition and schema preparation	11-16	Identify required source tables, define replication boundary,	Supports the first implementation increment by making

		and prepare the reporting database	the required data available
Initial replication and ETL workflow setup	14-18	Establish controlled extraction, loading, and manual rerun procedures	Provides the first usable data-refresh process for early report iterations
Iteration Cycle 1: early report reconstruction	16-20	Reconstruct selected reports at backend level and perform first parity checks	First implementation increment using the full analysis-design-build-validate loop
Iteration Cycle 2: rule refinement and additional reports	19-24	Extend report logic coverage, resolve first-wave discrepancies, and improve parity	Second increment that builds on feedback from the first validated modules
Portal workflow development and shared UI integration	20-26	Implement report search, parameter input, tabular viewing, and export workflows	Frontend functions are integrated in parallel with backend report increments
Iteration Cycle 3: expanded report reconstruction and regression checking	23-28	Add more report modules, recheck earlier modules, and reduce rule drift	Demonstrates repeated incremental extension with regression-aware validation
Controlled deployment and environment consolidation	25-30	Deploy validated modules, verify access control, and stabilise environment setup	Moves validated increments into controlled operational use
Source-to-replica quality checking and sync refinement	28-33	Strengthen portal-triggered sync support, quality re-	Extends the methodology beyond report logic into data-

		checking, and investigation workflow	quality assurance iterations
Scheduled ETL and administrative automation refinement	30-35	Introduce and refine scheduled ETL behaviour and related administrative controls	Represents a later operational increment after core parity work is sufficiently stable
Iteration Cycle 4: operational refinement and final validation pass	32-37	Revisit report behaviour, operational edge cases, and remaining mismatches	Final major refinement cycle before closure and handover preparation
Documentation, handover preparation, and final report consolidation	35-40	Consolidate technical documents, handover materials, and academic report inputs	Captures the outputs of all prior iterations into final closure artefacts

Table 3.2: Detailed Gantt-style project schedule across the 40-week internship period

3.5 Implementation Environment Requirements

The platform environment was specified at a practical level to support replication workloads, concurrent report queries, validation work, and controlled operational use. Table 3.3 summarises the high-level hardware considerations, while Table 3.4 summarises the software components and their roles within the platform. Together, these tables indicate that the project environment was designed around operational sufficiency for report reconstruction, testing, and delivery rather than around large-scale enterprise optimisation.

3.5.1 Hardware Requirements

At the hardware level, the platform required sufficient compute, memory, storage, network stability, and environment availability to support replication, API serving, and stakeholder validation activities. As summarised in Table 3.3, the hardware considerations were framed around dependable execution of replication and ELT tasks, responsive report generation, and stable access for iterative stakeholder review.

Resource	Requirement (high-level)	Justification
Compute	Multi-core CPU capacity	Supports replication and ELT tasks together with concurrent API request handling.
Memory	Sufficient RAM for database and service workloads	Supports SQL Server caching and in-memory processing during report generation.
Storage	Low-latency storage with growth headroom	Supports replicated tables, indexes, logs, and export artefacts with predictable query performance.
Network	Stable connectivity between source, replica, and services	Required for extraction, loading, and portal access; affects refresh time and report responsiveness.
Availability	Environment availability aligned to validation and operational use	Supports iterative validation and reduces disruption during reporting-intensive windows.

Table 3.3: Hardware requirements (high-level)

3.5.2 Software Requirements

At the software level, the platform depended on a relational database, a Python-based ELT environment, an API framework, a frontend framework, and version control tooling. As summarised in Table 3.4, these components collectively supported

the end-to-end workflow of data replication, report logic reconstruction, portal-based report delivery, and traceable change management.

Component	Role in the platform	Notes
Microsoft SQL Server	Hosts the replicated transactional schema	Provides relational storage for replicated tables and report queries.
Python	Implements replication, ELT scripts, and supporting utilities	Supports iterative development and operational scripting for extraction and loading.
SQLAlchemy + pyodbc	Database connectivity from Python	Provides a maintainable access layer to SQL Server via ODBC.
Microsoft ODBC Driver 17/18	ODBC connectivity to SQL Server	Driver version depends on host environment configuration.
FastAPI	Semantic/API layer for report endpoints	Exposes reconstructed report logic as RESTful services (FastAPI, n.d.).
React + Node.js	Reporting portal frontend	Implements report parameter inputs, tabular display, and export workflow (React, n.d.).
Nginx / web serving layer	Portal delivery and API proxy support	Supports frontend delivery and controlled routing in the deployed environment.
Git	Version control for code and documentation	Supports traceability of changes to report logic and validation artefacts.

Table 3.4: Software requirements (high-level)

3.6 Summary

This chapter presented the iterative methodology used to reconstruct vendor-aligned report logic, execute replication and ELT processes, and validate report-level parity through comparison, reconciliation, and supporting operational checks. Chapter

4 presents the analysis and design artefacts that operationalise this methodology for the Marrybrown Sales and Payment Analytics Platform.

Chapter 4

ANALYSIS AND DESIGN

4.1 Introduction

This chapter presents the analysis and design of the Marrybrown Sales and Payment Analytics Platform. The chapter translates the problem statement and project objectives into concrete requirements and design artefacts, covering system analysis, system requirements, current system analysis, and detailed system design. The academic basis for the design choices, including replication-first traceability, ELT, semantic/API layering, reconciliation, and iterative delivery, has been established in Chapter 2, while Chapter 3 described the methodology used to implement and validate the platform.

4.2 System Analysis

4.2.1 Case Study Context (Continuity Reporting for Sales and Payments)

The case study concerns sales and payment reporting in a large-scale food and beverage (F&B) retail setting where transaction processing and reporting access are provided through an externally managed POS environment and reporting portal. Continuity reporting is operationally important for Finance users, particularly for reconciliation and month-end closing, and for Operations users, particularly for outlet-level monitoring and exception review. In this context, the core system problem is that report access is mediated through vendor-managed interfaces and exports, which can constrain ad hoc investigation and introduce operational risk when the portal is unavailable or slow to respond.

4.2.2 Stakeholders and Role-Based View

Stakeholders are represented in a role-based manner to clarify reporting needs without asserting an official organisation chart.

Stakeholder group	Primary activities	Reporting needs (examples)
Finance users	Reconciliation, month-end closing, audit preparation	Payment breakdowns, voucher redemption visibility, return and cancellation checks, export-ready report outputs
Operations users	Outlet performance monitoring, operational oversight	Outlet-level report access, channel-specific sales views, exception visibility, report filtering by period and location
Internal technical team	Platform maintenance and enhancement	Traceability, rerun support, controlled report-logic updates, replication monitoring, operational administration

Table 4.1: Stakeholder groups and reporting needs (role-based view)

4.2.3 System Requirements Gathering Techniques

Requirements were derived through document analysis of vendor portal reports and exported files, together with stakeholder feedback and clarification sessions with Finance and Operations users. This approach is consistent with a black-box reporting environment in which proprietary report rules must be inferred from observable inputs and outputs rather than from a complete vendor specification. No survey instruments or formal interview claims are made; instead, the emphasis is on traceable artefacts such as exports, parameter settings, expected totals, and user-accepted report behaviour.

4.2.4 Use Case Diagram

The principal interactions between user roles and the platform can be summarised through a use case view. As shown in Figure 4.1, Finance and Operations users interact with the platform through six core use cases: access the report catalogue, select a report, apply report parameters, query the report output, review tabular results and totals, and export report outputs for reconciliation or operational follow-up. The internal technical team supports the platform through four operational use cases: execute replication refreshes, monitor runs and logs, rerun failed boundaries, and update report logic in a controlled manner. This use case view provides an analysis-level summary of the actor-system interactions before the more detailed design discussion in later sections.

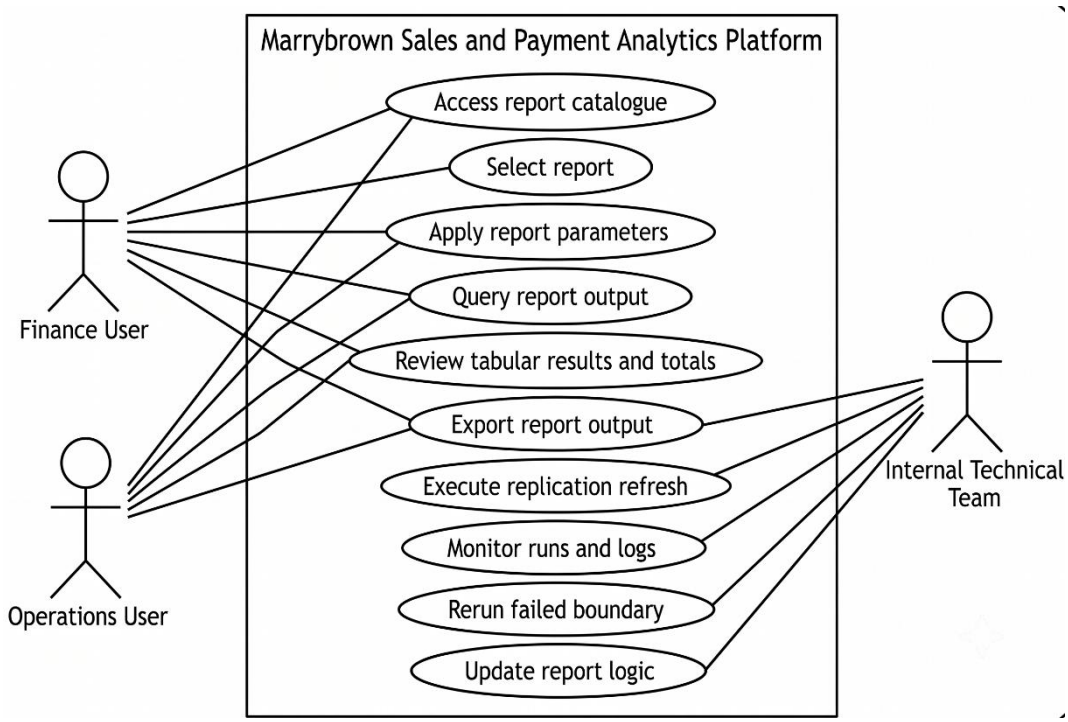

Figure 4.1: Use case diagram for the Marrybrown Sales and Payment Analytics Platform (illustrative)

4.2.5 User Requirement Outcomes

The requirement-gathering activities produced more specific outcomes than a general need for report access. For Finance and Operations users, the analysis indicated

that the platform had to preserve the practical behaviour of the vendor portal: a familiar report catalogue, parameter-driven retrieval, consistent option sets for shared parameters, exact tabular outputs for checking, and exportable results that could be used directly in reconciliation and month-end workflows. The findings also indicated that continuity reporting was prioritised over exploratory analytics, which is why detailed tables, totals, and export functions are treated as core interface outcomes rather than optional features.

For the internal technical team, the main outcomes concerned maintainability, traceability, and controlled operations. The platform had to preserve source-aligned replicated data, support vendor-equivalent rule implementation through SQL queries executed by the semantic/API layer, reuse common parameter and export logic across reports where possible, and provide controlled rerun, monitoring, and refresh support when replication or validation issues occurred. These outcomes provide the practical basis for the functional and non-functional requirements formalised in the next section and for the design decisions presented later in this chapter.

4.3 System Requirements

4.3.1 Functional Requirements (FR)

The functional requirements are summarised in Table 4.2.

ID	Functional requirement
FR1	The system shall replicate relevant sales and payment datasets from the external POS environment into a company-owned SQL Server reporting database through controlled refresh processes.
FR2	The system shall reconstruct and generate company-required sales and payment reports based on the replicated datasets.
FR3	The system shall provide report filtering by report-specific parameters, at minimum date range and outlet scope where applicable.

FR4	The system shall provide tabular report outputs, totals or subtotals where required, and export support for reconciliation workflows.
FR5	The system shall expose report logic through an API layer so that the frontend remains decoupled from underlying database structures.
FR6	The system shall support controlled refresh-related administrative actions, including rerun or follow-up handling for reporting data updates.

Table 4.2: Functional requirements for the platform

4.3.2 Non-Functional Requirements (NFR)

The non-functional requirements are summarised in Table 4.3.

ID	**Non-functional requirement**
NFR1	Data fidelity and traceability: report outputs shall remain traceable to replicated source records and support reconciliation checks.
NFR2	Availability: the platform shall provide an internal reporting path when vendor portal access is disrupted or operationally inconvenient.
NFR3	Performance (operational usability): report responses shall be delivered within an acceptable time for reporting-intensive workflows such as month-end closing.
NFR4	Security: the platform shall operate as a read-only reporting environment and implement appropriate access control.
NFR5	Maintainability: report logic shall be structured to support iterative refinement when edge cases, missing data conditions, or rule changes are discovered.
NFR6	Auditability: refresh activity, validation outcomes, and report-serving behaviour shall be sufficiently observable to support operational follow-up.

Table 4.3: Non-functional requirements for the platform

4.3.3 Constraints and Assumptions

The platform was designed under several constraints and assumptions. Access to the external POS environment was restricted and read-only from the perspective of the reporting platform. The platform was intended for reporting continuity and internal operational access rather than write-back interaction with the vendor POS system. The refresh cadence was designed to support operational reporting needs rather than real-time transactional processing. Historical completeness was also dependent on approved replication boundaries and the treatment of late corrections. Design decisions in this chapter therefore prioritise traceability, repeatability, and controlled reporting operations over transactional immediacy.

4.4 Current System Analysis

The original reporting workflow relied on a vendor-managed reporting portal as the primary interface for sales and payment reports. Users typically selected a report, applied parameters such as outlet and date range, and exported results for reconciliation activities. This workflow constrained ad hoc analysis because users could not freely query underlying transactional records, and operational reporting could be disrupted if portal access degraded during reporting-intensive periods. Issue resolution was also dependent on vendor support processes, which did not always align with internal reporting timelines.

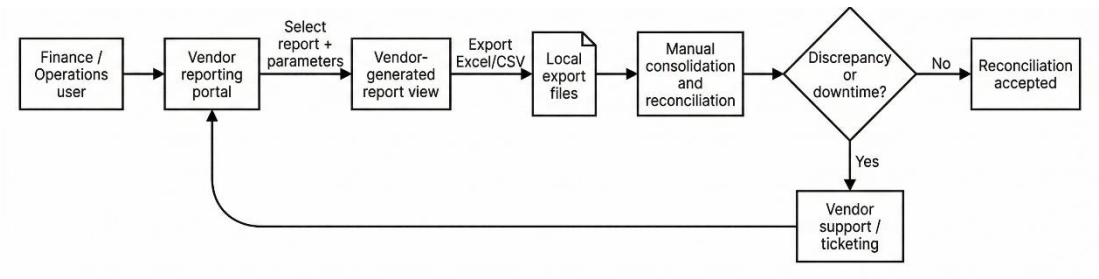

Figure 4.2: Current vendor-dependent reporting workflow (simplified)

4.5 System Design

4.5.1 System Architecture

The platform is organised as an end-to-end analytical flow comprising six connected stages: vendor-managed POS data sources, selective replication and ELT, a company-owned SQL Server replica, a semantic/API layer, a reporting portal, and Finance and Operations users. This structure preserves source-aligned data in the replica while centralising report rules, validation logic, and delivery behaviour at the FastAPI service layer, consistent with the replication and semantic-layer rationale discussed in Chapter 2.

As shown in Figure 4.3, the architecture begins with vendor-managed transactional data and vendor report outputs, which feed a selective boundary-based replication and ELT process into the company-owned SQL Server replica. The FastAPI semantic layer then applies report rules and validation over the replicated data before the reporting portal presents report lists, parameter-driven retrieval, tabular outputs, and export functions to Finance and Operations users. In this way, the architecture delivers continuity reporting through a company-controlled path while preserving separation between data acquisition, storage, business-rule processing, and presentation.

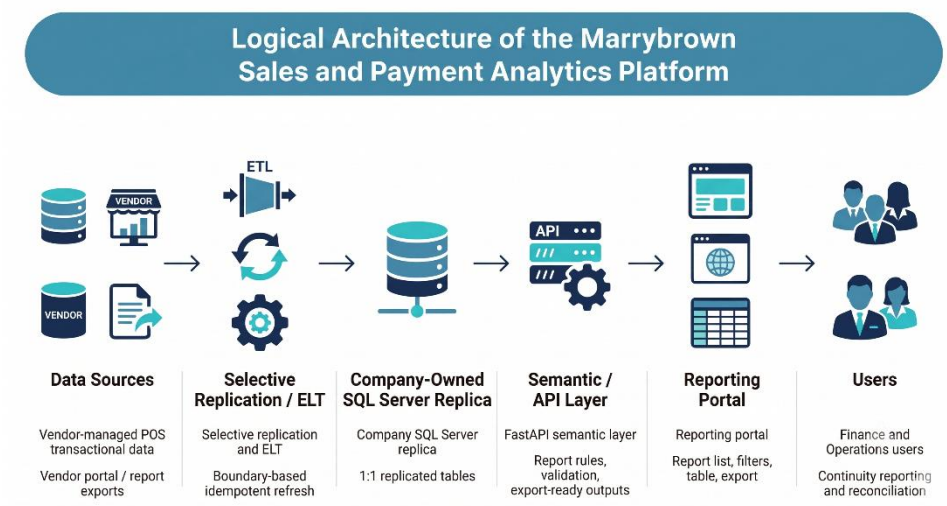

Figure 4.3: Logical architecture of the analytics platform

4.5.2 Component Explanations

Table 4.4 summarises each component's responsibility and key design considerations. These component-level responsibilities are aligned with the literature reviewed in Chapter 2, particularly the discussions on replication-based analytical stores, ELT workflows, semantic layers, service interfaces, and operational reporting portals.

Component	Responsibility	Key design considerations
External POS environment	Source of transactional records and baseline report behaviour	Access constraints; parameter consistency; late corrections
Replication / ELT processes	Extract and load source data into the replica	Idempotent refresh boundaries; logging; rerun controls; quality checks
SQL Server replica	Store 1:1 replicated tables and support report queries	Traceability; indexing strategy; retention boundaries; reporting-readiness
FastAPI semantic layer	Encapsulate report rules and expose stable report endpoints	Versioned business rules; parameter validation; consistent metric definitions; controlled access
Reporting portal	Provide internal report access, export, and limited administrative operations	Usability during reporting periods; consistent filtering; totals/subtotals; operational visibility

Table 4.4: Component responsibilities and design considerations

4.5.3 Report Coverage Overview

The platform design was not limited to a single dashboard or a small number of isolated queries. It was structured to support a formal implementation scope comprising 19 sales and payment reports, together with three additional customised

reports requested during implementation. At design level, these reports can be grouped into several functional categories, as summarised in Table 4.5 and Table 4.6. Detailed per-report specifications are provided in Appendix A.

Report category	**Formal implementation/business-scope reports**	**Main design implications**
Sales and exception control	Sales Return Report; Sales Cancelled Report; DELETED Items Report; Pickup & Declaration Report; [SOK] Each Kiosk Transaction Report	Requires transaction-status handling, traceability to sale-level records, and exact tabular outputs for checking and audit follow-up
Payment and voucher reconciliation	Payment type (All Payment); MB Cash Voucher (with Barcode) Redemption Report; MB Staff E - Voucher RM 20 & MB CASH VOUCHER RM10 (with Barcode) Redemption Report	Requires payment-to-sale linkage, voucher normalisation, and export fidelity for reconciliation workflows
Delivery and ordering channels	Sale Delivery (By Sales Type) Ex Tax Calculation; Delivery - FoodPanda, Grabfood, ShopeeFood; Foodpanda Sales; Foodpanda Discount; Mobile Ordering Sales	Requires channel classification, period-based filtering, and consistent grouping rules across delivery and ordering sources
Product, stock, and cost analysis	Product Mix Report; Product Mix with modifier without ETL; Stock Variance Report (Latest); Xilnex - Monthly	Requires item-level granularity, reference-data support, and careful handling of derived values such as cost,

	Checking - COGS by Item (By Sales Type)	quantity, and profit-related measures
Service and operational timing	Average SOS Report (New)	Requires timing-oriented aggregation over KDS-backed sales activity and store-level operational performance interpretation
Discount monitoring	Discount Remark Report	Requires discount-level traceability, remark handling, and alignment between discount records and sales context

Table 4.5: Formal implementation/business-scope report groups and design implications

Additional customised report	**Reporting purpose**	**Design relationship to the shared platform**
Roblox Free Chicken Burger Combo Sales	Tracks a company-requested campaign report for a specific promotional sales scenario	Reuses the same parameter, query, export, and portal delivery pattern as the formal report set
Voucher Campaign & Reward Sales	Tracks voucher-campaign and reward-related sales activity requested during implementation	Extends the shared semantic/API design with campaign-focused grouping and output rules
Promotion Item Additional Purchase Report	Analyses additional-purchase behaviour for configured promotion items	Reuses the shared reporting workflow while extending the design with configurable promotion-item categorisation

Table 4.6: Additional customised reports and their relationship to the shared platform design

4.5.4 Data Engineering and API Design

4.5.4.1 Data Sources and Replication Boundary

The primary data source is the vendor-managed POS transactional database, which is accessed for read-only replication into the company-owned SQL Server environment. At a logical level, the source data required by the report set can be grouped into transaction-header records, line-item records, payment or tender records, and supporting master or reference data. Transaction-header data provides the reporting grain and status context for each sale, line-item data supports item-level analysis and quantity or amount breakdowns, payment data supports payment-type reporting and reconciliation, and master or reference data such as outlet and item attributes is required to interpret and group transactional records consistently.

Replication boundaries were defined report by report by tracing required output columns and business rules back to the minimum set of source tables needed for reconstruction. In the implemented environment, transactional replication prioritised the operational history required by the report set, while smaller reference datasets were retained more broadly because they were required to interpret transactions consistently across reporting periods. This selective but fidelity-preserving boundary supports operational reporting windows while allowing late corrections to be refreshed during subsequent runs.

4.5.4.2 Refresh Cadence and Idempotent Load Pattern

The ELT workflow was designed to refresh operational data in a controlled manner and to account for late corrections such as voids, adjustments, and return postings. The implemented design supports both portal-triggered refresh activity for operational follow-up and scheduled ETL refreshes for routine reporting maintenance. This design operationalises the ELT and idempotent-load principles discussed in Chapter 2, where controlled load boundaries and deterministic reruns are identified as the basis for repeatable refresh cycles.

Within each refresh boundary, existing rows for the affected reporting window are removed from the replica and then re-inserted from a fresh extract. This design ensures that rerunning the same boundary yields a consistent replicated state and supports recovery when extraction or load failures occur. As illustrated in Figure 4.4, the refresh and report-serving sequence can be described in two stages. During the refresh stage, the ELT process extracts source transactions for a defined boundary window, refreshes the corresponding rows in the SQL Server replica, and performs post-load checks. During the report-serving stage, the portal submits a report request with user-specified parameters, the API queries the refreshed replica and applies semantic rules, and the resulting rows, totals, and export-ready output are returned to the portal. This separation clarifies that replication and report serving are linked operationally but remain distinct responsibilities within the overall platform design.

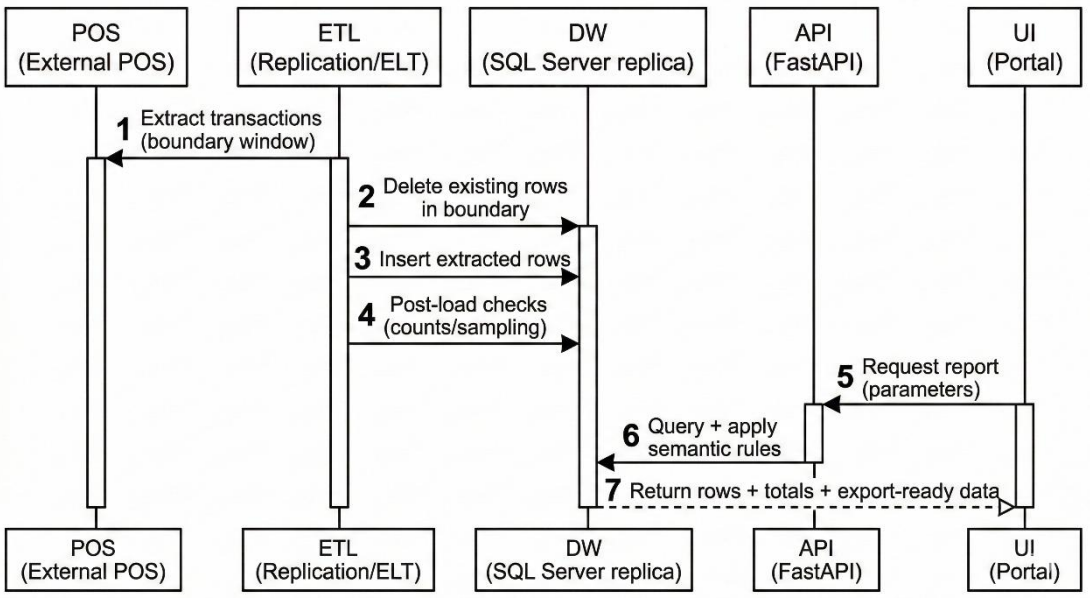

Figure 4.4: Boundary-based refresh and report-serving sequence (illustrative)

4.5.4.3 Operational Controls, Logging, and Data-Quality Checks

Operational controls were designed to support reliability, recovery, and reporting trustworthiness. At design level, these controls include logging of refresh outcomes, post-load checking, source-to-replica quality verification for selected reporting windows, and rerun capability for specific boundaries. These controls ensure that the replicated state can be monitored and trusted before it is used for report

reconstruction. The detailed implementation and operational handling of these controls are presented in Chapter 5.

4.5.4.4 API Endpoint Pattern and Semantic Layer Responsibilities

The API layer acts as a semantic layer that encapsulates reconstructed business rules and provides a stable interface to the reporting portal. Report endpoints follow a consistent pattern built around date-range parameters, outlet-scope parameters where relevant, and report-specific optional filters. For each endpoint, the design specifies required and optional parameters, the output schema, totals or subtotals where required, formatting rules, and the expected validation approach.

This design reflects the semantic-layer and service-interface concepts discussed in Chapter 2. Replicated data are retained close to source structure for traceability, while report-specific business rules, derived fields, and delivery behaviour are centralised at the API layer. The API therefore decouples report consumption from underlying storage design and keeps report logic more maintainable, versioned, and auditable.

4.5.4.5 Security Considerations

The platform is designed as a read-only reporting environment. Authentication and authorisation are applied to ensure that only authorised users can retrieve reports or access role-restricted administrative functions. Query parameters are also validated so that report access remains bounded by the intended workflow and organisational reporting policy.

4.5.4.6 Report Reconstruction Overview

Report reconstruction follows a repeatable workflow: define report inputs and filters, identify contributing replicated datasets, implement vendor-aligned business rules at the semantic layer, aggregate and format outputs into a stable schema, and validate parity against vendor outputs. This workflow is shared across the formal report set and the additional customised reports, even though each report has its own business rules and output structure.

At Chapter 4 level, the main design point is that the platform supports multiple report types through a shared semantic/API pattern rather than through unrelated one-off queries. Reports differ in grouping logic, filters, and output fields, but they follow the same overall design principles of traceable replicated data, parameter-driven retrieval, stable tabular output, and export-oriented delivery. Detailed implementation examples and validation journeys are presented in Chapter 5, while detailed per-report specifications are provided in Appendix A.

4.5.5 Database Design

The data design adopts a 1:1 replication strategy, in which the company-owned database mirrors relevant transactional tables from the external POS environment. This strategy prioritises fidelity and traceability to support reconciliation and parity validation. It is also aligned with the schema-on-read rationale discussed in Chapter 2, where duplicated transactional data are retained close to source structure for traceability, while report-specific calculations and derived logic are handled separately at the semantic/API layer.

As illustrated in Figure 4.5, the conceptual model centres on sales headers, sales items, payments, and supporting reference entities such as location and item master data. This model is intended to show the main reporting entities and relationships at a conceptual level rather than to expose vendor-specific physical table names.

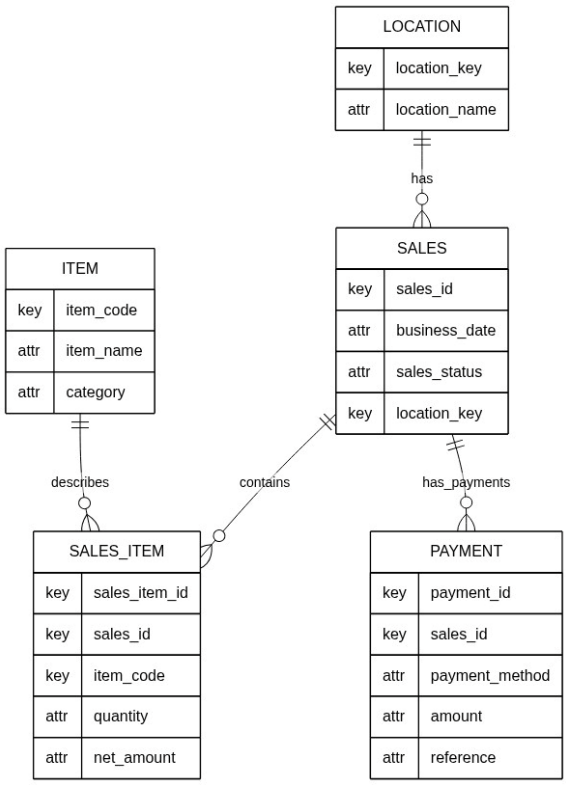

Figure 4.5: Simplified conceptual data model for sales and payment reporting (illustrative)

Table 4.7 summarises the main conceptual entities required by the report set. Physical table names and column names in the replicated schema are vendor-specific; the design intent is to preserve a 1:1 representation while documenting join paths and key attributes used in report logic.

Entity	**Purpose in reporting**	**Representative attributes (conceptual)**
Sales (header)	Defines the reporting grain for sales totals, date scoping, and status handling	Sale identifier; business date; outlet/location key; sales status; totals
Sales item (line)	Supports item-level reporting, quantity analysis, and return or adjustment detection	Sale identifier; item code or name; quantity; net amount; tax fields

Payment	Supports payment type breakdown, voucher handling, and reconciliation	Sale identifier; payment method; amount; device or reference fields
Location	Maps location identifiers to outlet names and outlet-scope filtering	Location key; location name; hierarchy attributes where applicable
Item master	Provides item metadata and grouping attributes	Item code or name; category; group; division attributes

Table 4.7: Conceptual data dictionary (summary)

4.5.6 Business Rule Reconstruction for the Formal Report Set

Although the replicated database preserves source-aligned transactional structures, the business rules for the formal sales and payment report set are reconstructed and applied within the semantic/API layer rather than embedded in the storage layer. This separation allows the replica to remain close to source structure for traceability, while report-specific logic is centralised above the database layer for maintainability and auditability.

The semantic/API design supports the report set through a common service pattern. Each report is exposed through a dedicated endpoint, but follows a shared response and delivery approach based on flat-list outputs, common parameter parsing behaviour, and consistent export handling. Shared parameter logic is reused where reports expose the same option sets, while report-specific extensions allow additional filters such as item division, group name, channel classification, campaign criteria, or department where required.

In this project, business-rule reconstruction refers to inferring vendor-aligned reporting rules from observed portal outputs and implementing those rules as SQL queries executed by FastAPI over the 1:1 replicated data. The semantic/API layer therefore applies report-specific selection, grouping, aggregation, and formatting logic without changing the underlying replicated structure. Appendix A documents the

detailed per-report specifications. Within the main design chapter, the emphasis is therefore on the shared reconstruction pattern and service-layer responsibilities rather than repeating each individual report specification in full.

4.5.7 Interface Design

The portal is designed to mirror the existing user workflow for continuity reporting while providing more controlled internal access to report selection, parameter entry, report retrieval, and export. The primary interaction is: authenticate, select report, set parameters, view results with totals or subtotals where relevant, and export outputs for reconciliation or operational review. In addition to ordinary report retrieval, the portal design also accommodates role-restricted administrative views such as user administration and refresh-related operational oversight.

As illustrated in Figure 4.6, the report-list view is designed to support report discoverability and selection through a structured catalogue of available reports. This interface allows users to identify the required report quickly, recognise its reporting purpose from the accompanying description, and navigate into the relevant reporting screen without relying on vendor-managed menus.

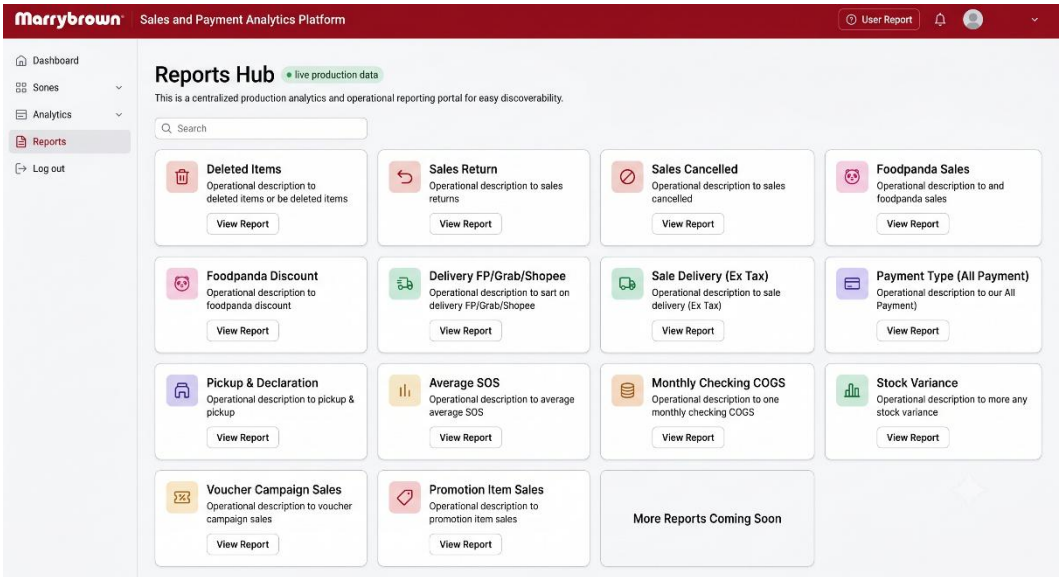

Figure 4.6: Report list interface (illustrative)

As illustrated in Figure 4.7, the reporting view combines parameter controls, query execution, summary indicators, tabular output, and export actions within a single workflow. This design enables users to specify report parameters, retrieve vendor-aligned outputs, inspect key totals or subtotals, and export the result for reconciliation or further review. Rather than prioritising chart-heavy dashboards, the interface emphasises summary indicators and detailed tabular presentation because the project objective is continuity reporting and parity-aligned operational access rather than exploratory visual analytics.

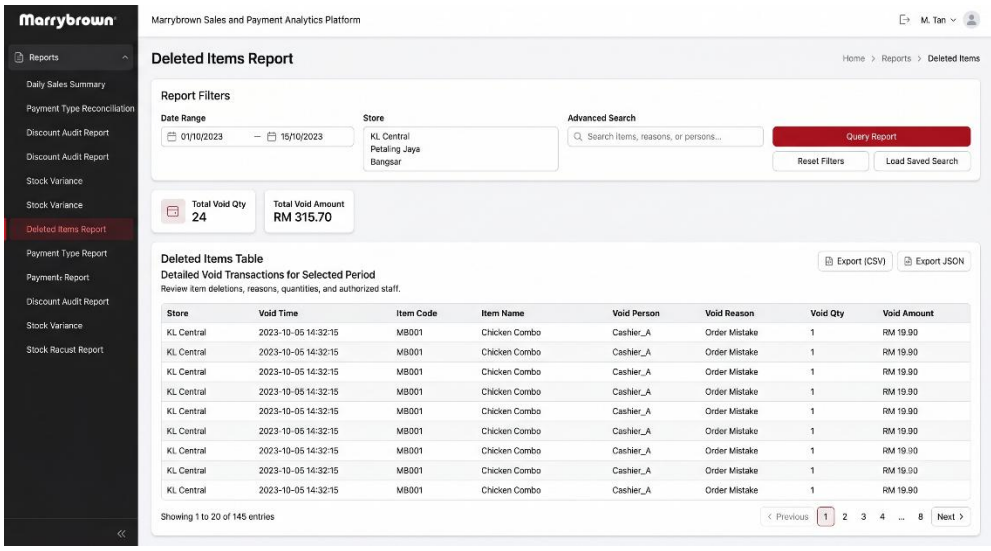

Figure 4.7: Reporting interface (illustrative)

Figure 4.8 provides a simplified workflow view of the overall portal interaction, linking login, report discovery, report selection, parameter entry, result inspection, export, and downstream reconciliation. While Figures 4.6 and 4.7 show specific interface states, Figure 4.8 summarises the intended end-to-end navigation path so that the relationship between report selection and report consumption can be understood at a glance.

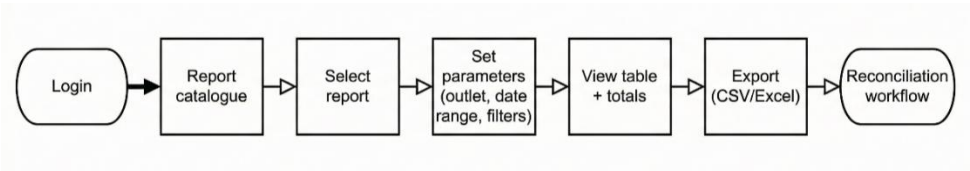

Figure 4.8: Portal navigation and report workflow (illustrative)

Because the delivered system also included role-restricted operational administration, the interface design was not limited to ordinary report-consumption pages alone. As illustrated in Figure 4.9, the administrative interface design groups Data Sync, Automation Control, and User Management into a controlled support surface so that authorised users can handle replication follow-up, automation oversight, and access administration within the same portal environment. This figure should be presented at overview level and should exclude sensitive operational details such as internal hostnames, user emails, or private infrastructure identifiers.

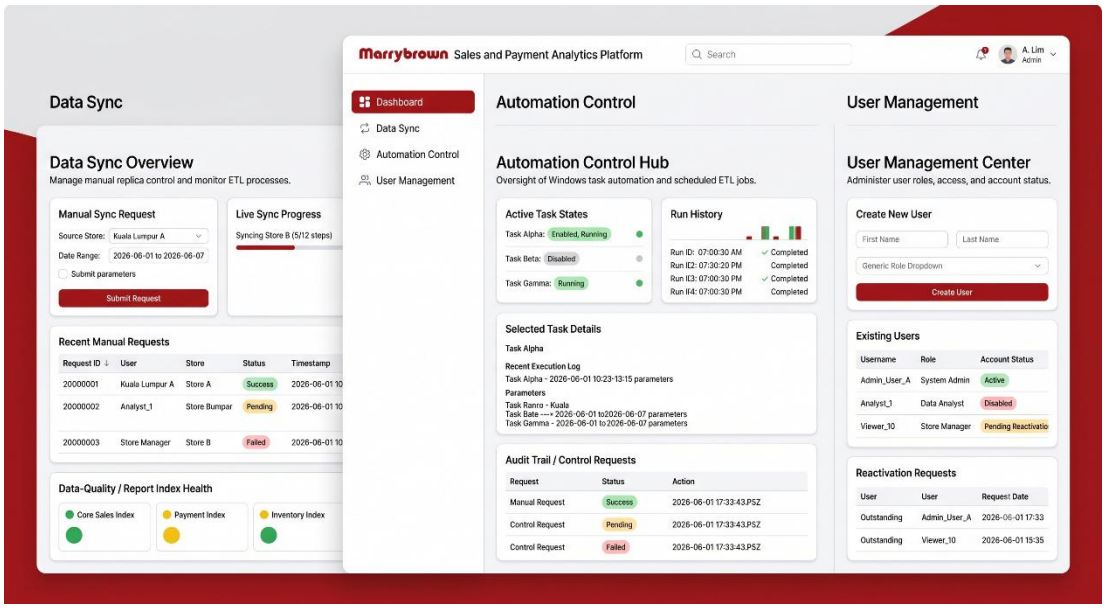

Figure 4.9: Administrative interface overview (illustrative)

Interface design requirements prioritise consistent parameter controls across reports, tabular presentation aligned to expected output structures, and export outputs that support downstream reconciliation. These priorities are consistent with the workflow and interface considerations for operational reporting portals discussed in Chapter 2. Portal design details were refined iteratively based on stakeholder feedback during development and validation cycles.

4.5.8 Summary

This chapter presented the system analysis, requirements, current system workflow, and the design of the Marrybrown Sales and Payment Analytics Platform. The design emphasises traceability, report reconstruction, parity-oriented delivery, and continuity reporting through replication-first data management, a semantic/API layer for business rules, and a portal interface that supports parameter-driven retrieval and export-based checking. The chapter also outlined how the shared design supports the formal report set and the additional customised reports. Chapter 5 presents the implementation and testing work carried out on the basis of this design.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Introduction

This chapter presents the implementation and testing of the Marrybrown Sales and Payment Analytics Platform. Whereas Chapter 4 described the analysis and design of the platform, this chapter records how the implemented platform was developed across the reporting database and ETL layer, the FastAPI semantic/API layer, and the React-based portal layer. It also documents the validation and testing activities used to confirm report accuracy, user-facing behaviour, and internal processing reliability.

The chapter focuses on the completed implementation rather than on design intent alone. Accordingly, attention is given to the reporting-database replication process, report reconstruction workflow, frontend report delivery, administrative functions, report implementation coverage, and the testing approach used to evaluate the retained report set and additional customised reports. To keep the chapter academically appropriate and public-safe, implementation details are presented at technical level without exposing credentials, secret configurations, or operationally sensitive access instructions.

5.2 System Development

5.2.1 Development Overview

The implemented platform was developed as a multi-layer internal reporting system intended to reduce operational dependence on a vendor-managed reporting portal while preserving report traceability and validation discipline. The completed

implementation consists of three primary layers. First, selected source-system data were replicated into a company-managed Microsoft SQL Server reporting database through controlled ETL processes. Second, report logic was reconstructed in a FastAPI backend that acted as a semantic/API layer over the replicated data. Third, a React-based reporting portal was implemented to provide authenticated report access, filtering, export functions, and selected administrative tools.

The development process followed the iterative and incremental methodology described in Chapter 3. This was necessary because several report modules could not be treated as straightforward query transcription tasks. In practice, report development involved repeated cycles of source-data tracing, semantic rule reconstruction, mismatch investigation, validation against vendor outputs, and refinement of the reporting workflow in both backend and frontend layers.

5.2.2 Reporting Database and ETL Development

The reporting-database implementation centred on a company-managed Microsoft SQL Server environment that preserved selected source tables through a 1:1 replication strategy. This replica design was chosen to retain source fidelity for parity validation and reconciliation, while allowing report logic to be rebuilt independently of the external portal. Rather than flattening all logic into the database layer, the implementation preserved transactional and reference data close to source structure and left report-specific grouping, derivation, and presentation logic to the semantic/API layer.

The ETL implementation did not rely on a single loading path. Instead, it evolved into a controlled replication model with separate but coordinated operational modes. Manual utilities were retained for broad baseline loading, historical backfill, and targeted recovery work, especially for dated tables that required replay across defined time windows. In parallel, a staged daily ETL orchestrator was implemented for routine dated-table maintenance and reference-data refreshes. This staged design used boundary-based replacement logic so that repeated execution of the same

reporting window could safely refresh the replica without introducing duplicate or drifting rows.

In the final implemented state, the ETL layer also supported portal-triggered manual sync requests and scheduled automation. Portal-triggered sync allowed authorised administrators to request dated-table or reference-table refreshes from the portal without executing ETL commands directly on the database host. Scheduled ETL automation was also implemented as an operational feature for unattended daily dated refreshes and nightly retained-reference refreshes, with administration performed through the platform's automation-management workflow. This arrangement gave the platform both controlled manual recovery capability and operational automation capability, without requiring ordinary report users to interact with the ETL layer directly.

5.2.3 Backend API Development

The backend was implemented using FastAPI and functioned as the semantic/API layer of the platform. Its principal role was to reconstruct vendor-aligned report logic over the replicated database and expose the resulting report outputs through stable, parameter-driven endpoints. This design avoided direct frontend dependence on raw database structures and made it possible to keep report logic versioned and maintainable within the backend layer.

Backend development involved several recurring tasks across report modules. These included request-parameter definition, outlet and date-range filtering, shared handling of sales-status or payment-status filters where relevant, structured row output, and export-oriented response shaping. Reports that required grouped behaviour in the portal also needed backend outputs that could be normalised reliably for frontend grouping, subtotal rendering, advanced search, and export generation. In this way, the backend was not merely a database wrapper; it became the primary location where report-specific business rules, derived fields, and stable response contracts were implemented.

The backend layer also supported non-report operational functions required by the completed system. These included authentication, admin-managed user functions, portal-triggered sync control, automation-management support, and quality-check workflows linked to selected sync operations. Although these supporting features were not the central academic contribution of the project, they were necessary to make the implemented platform operationally usable in a real company environment.

5.2.4 Frontend Portal Development

The frontend portal was implemented using React and was designed to provide authenticated internal access to reports through a workflow-oriented rather than dashboard-oriented interface. The implemented report experience followed a common pattern: users sign in, access the report catalogue, select a report, define parameters, query the report, review the resulting table and totals, and export the output where needed. This structure was intentionally aligned with the practical behaviour expected by Finance and Operations users, especially for reconciliation and operational checking tasks.

Frontend development relied on shared report architecture rather than isolated page-specific implementations. A report registry, shared page shell, shared filters panel, grouped-table component, export toolbar, and supporting hooks were reused across report modules. This reduced duplication and made it feasible to add new formal-scope reports and additional customised reports through a consistent frontend pattern. The report-list view, report-query view, grouped-table behaviour, advanced search support, and export actions were therefore developed as reusable platform capabilities rather than as unrelated page fragments.

This user-facing reporting pattern is reflected in Figure 5.1 through a sanitised composite view that combines the Reports Hub with a representative report-query page. The figure represents the actual portal flow retained in the implementation: users begin from the catalogue of available reports, move into the selected module to define the required parameters, and then review grouped output through the shared query

interface. Read in this way, the figure supports the main implementation point of this subsection, namely that the portal was developed as a consistent report-execution surface rather than as a collection of unrelated dashboard screens.

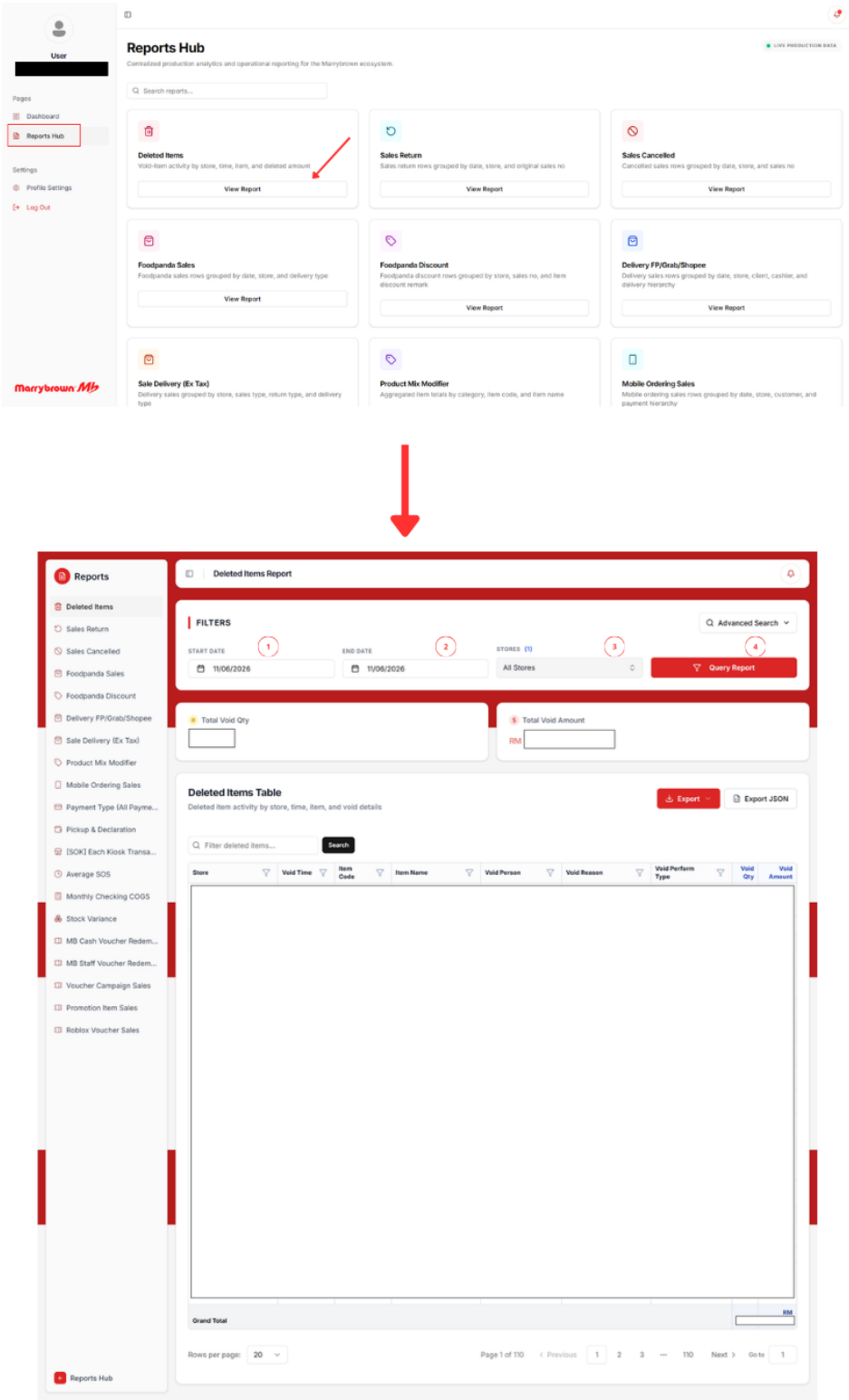

Figure 5.1: Reports Hub and representative report-query interface

The implemented portal also included role-restricted supporting features beyond ordinary report retrieval. Admin-only views were developed for data sync, automation control, and user management. These pages extended the platform beyond passive report viewing and reflected the fact that the delivered system functioned as an internal reporting platform with controlled operational administration, not only as a collection of static report pages.

5.2.5 Administrative and Operational Functions

Several administrative and operational functions were implemented to make the platform sustainable for real use after report delivery. The first was admin-managed user control under the authentication model, where normal user access to report pages was separated from admin-only access to management functions. The second was the portal-triggered manual sync workflow, through which administrators could request sales-window or reference refreshes and review request history and progress. The third was automation management, where authorised administrators could review scheduled-task status and request enable, disable, or schedule-change actions for routine ETL automation.

An additional operational enhancement was the data-quality checking layer for portal-triggered replication. This feature allowed administrators to review source-to-replica quality outcomes for selected reporting windows, perform re-check actions, and trigger repair-oriented follow-up where required. This function was relevant academically because it showed that report continuity depended not only on endpoint implementation, but also on the trustworthiness of the replicated data used by those endpoints.

The administrative layer described above was implemented as three coordinated portal surfaces, each tied to a distinct governance responsibility. Figure 5.2 corresponds to user administration, where authorised staff review accounts and maintain role-controlled access. Figure 5.3 corresponds to data-sync administration, where replication requests, execution histories, and related follow-up actions are

managed from the portal. Figure 5.4 corresponds to automation administration, where scheduled ETL activity is reviewed and change requests are handled under controlled access. Considered together, these figures reinforce that the delivered platform supported governed internal operations in addition to ordinary report retrieval.

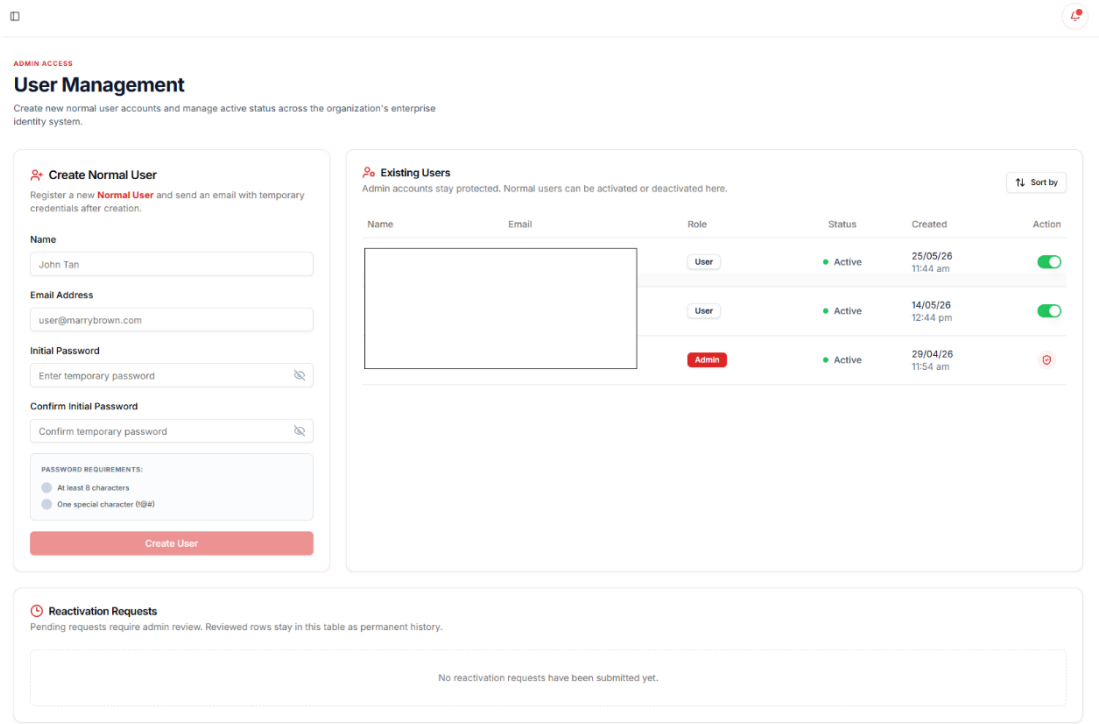

Figure 5.2: User Management page

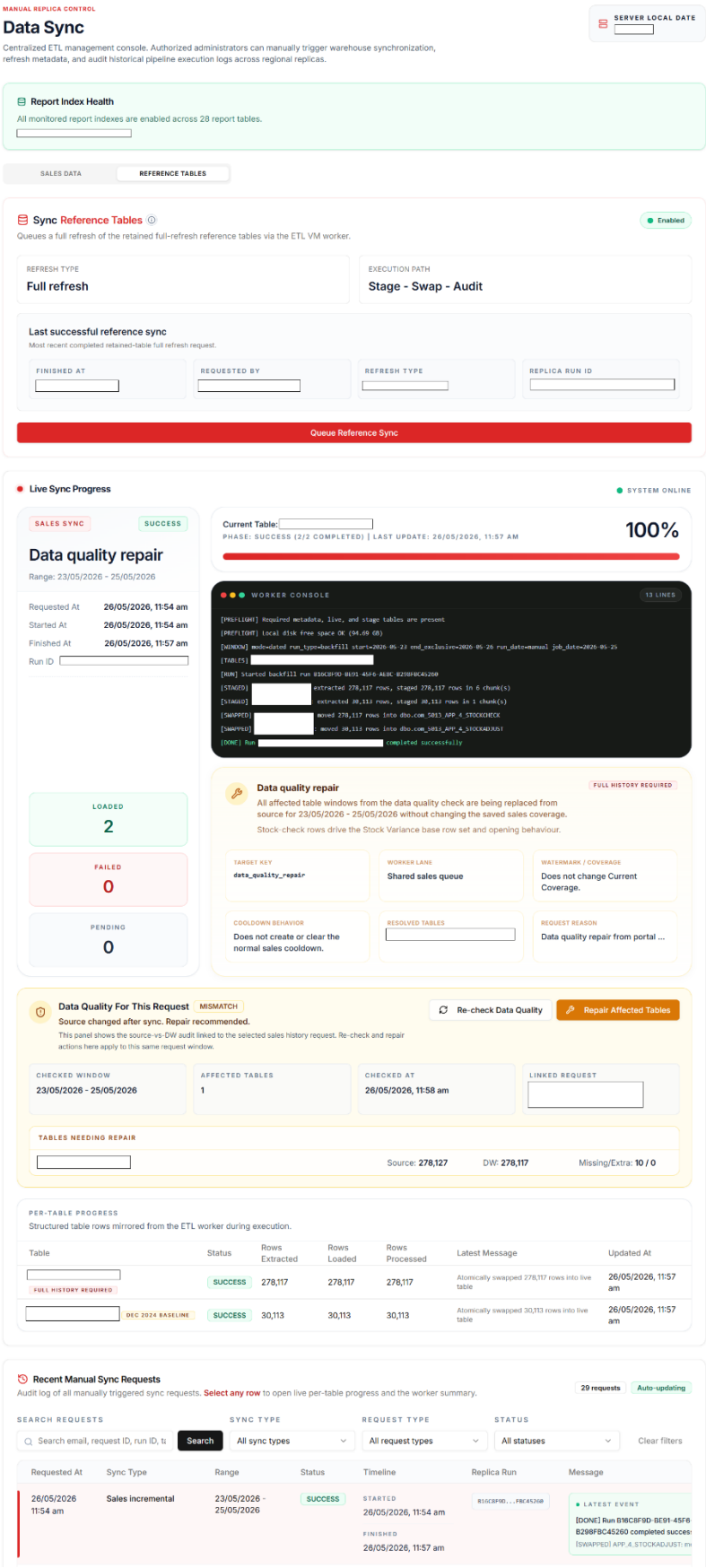

Figure 5.3: Data Sync page

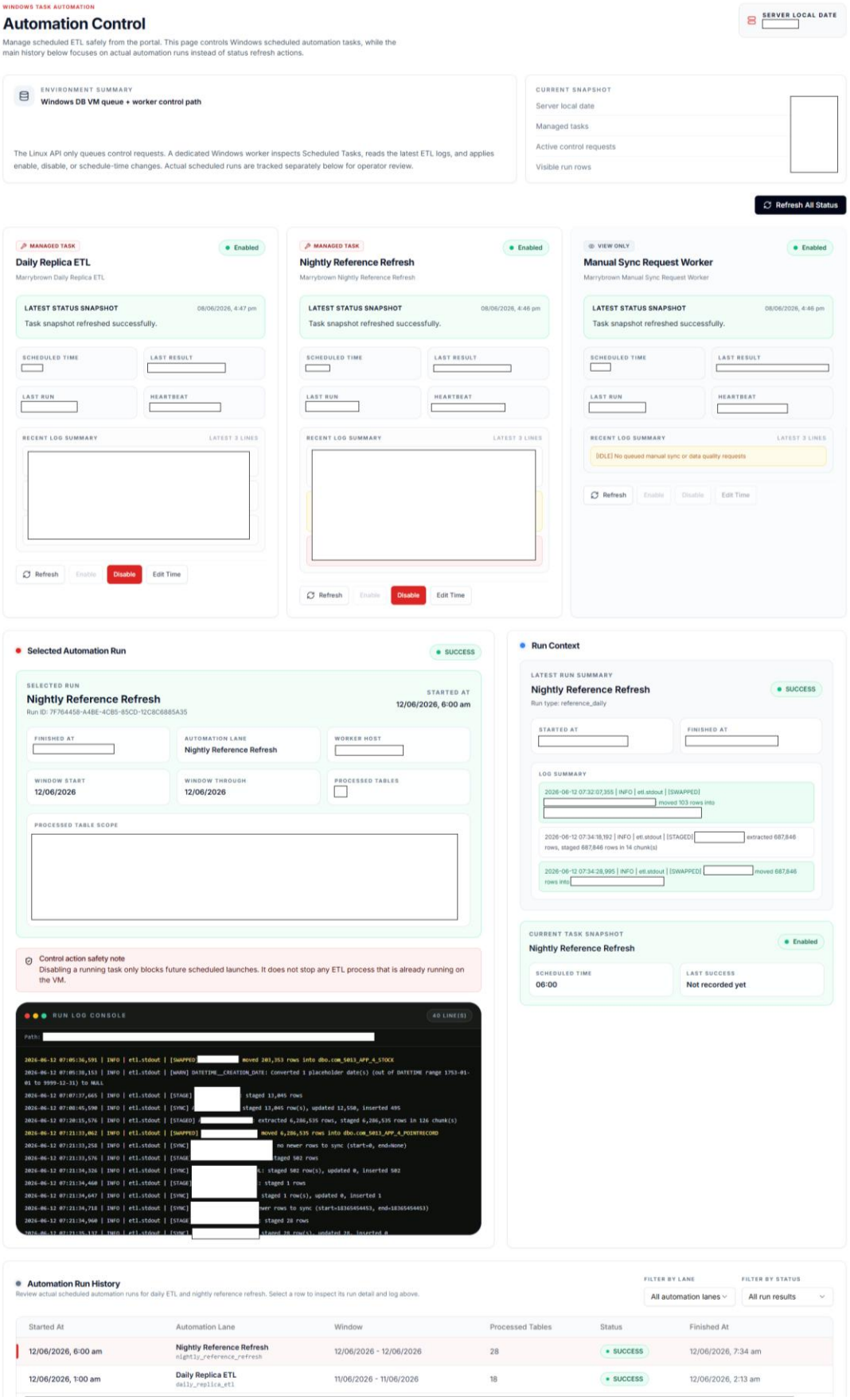

Figure 5.4: Automation Control page

5.2.6 Report Implementation Coverage

Within the formal implementation and business scope, the platform was developed to support 19 sales and payment reports. Of these, 16 reports were successfully implemented and retained in the delivered active reporting surface. Three reports were not fully closed within the project boundary and are documented as limitations rather than omitted outcomes. In addition to the formal 19-report scope, three additional customised reports were also developed based on company requests during implementation.

The retained formal-scope reports covered several reporting groups, including sales and exception control, payment and voucher reconciliation, delivery and ordering channels, and product-, stock-, or cost-related operational analysis. The additional customised reports extended the implemented platform beyond the original formal scope and demonstrated that the shared ETL, backend, and portal architecture could support company-requested report families without requiring a separate system. Detailed per-report specifications, statuses, and classifications are documented in Appendix A.

The three formal-scope reports that were not fully closed were Product Mix Report, Discount Remark Report, and Product Mix with modifier without ETL. These were not left unresolved due to a simple lack of page development. Rather, they remained unresolved after reverse engineering and parity validation because the remaining logic required for full closure, especially profit-related behaviour in the product-mix area, could not be isolated conclusively within the available project boundary.

5.3 Coding of the system's main functions/Process

5.3.1 Data Replication and Refresh Process

The main data-process function of the platform was the movement of selected source data into the reporting database in a form suitable for report reconstruction. This process followed a staged boundary-based refresh pattern. For dated-table maintenance, the ETL process identified a defined reporting window, extracted source rows for that window, staged the refreshed data, replaced the corresponding live rows in the reporting database, and then recorded run metadata and progress information. This approach ensured that rerunning the same reporting window remained deterministic and that failed or incomplete windows could be retried without depending on ad hoc manual cleanup.

This process was important because reporting mismatches were not caused only by query logic. In several cases, mismatches were associated with late source edits, historical incompleteness, or replica-state inconsistencies. The implemented refresh model therefore had to support not only normal data movement but also controlled replays and focused correction work. For broad historical loading, manual utilities were used for baseline and backfill operations. For routine maintenance, scheduled daily ETL and portal-triggered sync requests used the staged orchestrator path. This split between baseline loading and operational refresh was necessary to balance reliability, recoverability, and maintainability.

The implemented process also included quality-oriented follow-up for selected portal-triggered sync windows. After a successful sales-window refresh, a source-to-replica quality check could be launched for the same window. Where mismatches were identified, re-check or repair-oriented follow-up could be requested. This meant that the data process for the platform was not limited to extraction and loading alone; it also included mechanisms for checking whether the reporting replica remained trustworthy enough for subsequent report reconstruction.

This two-path refresh model is summarised in Figures 5.5 and 5.6. Figure 5.5 represents the retained reference-table refresh path, in which a full refresh request is initiated from the portal and then monitored through staged execution and per-table progress review. Figure 5.6 represents the dated sales-data refresh path, where a bounded sync request moves through queued execution, post-run data-quality review, and repair-oriented follow-up for the selected reporting window. Presented in this order, the figures support the subsection's argument that portal-triggered replication was implemented as a controlled and auditable administrative process rather than as an opaque background task.

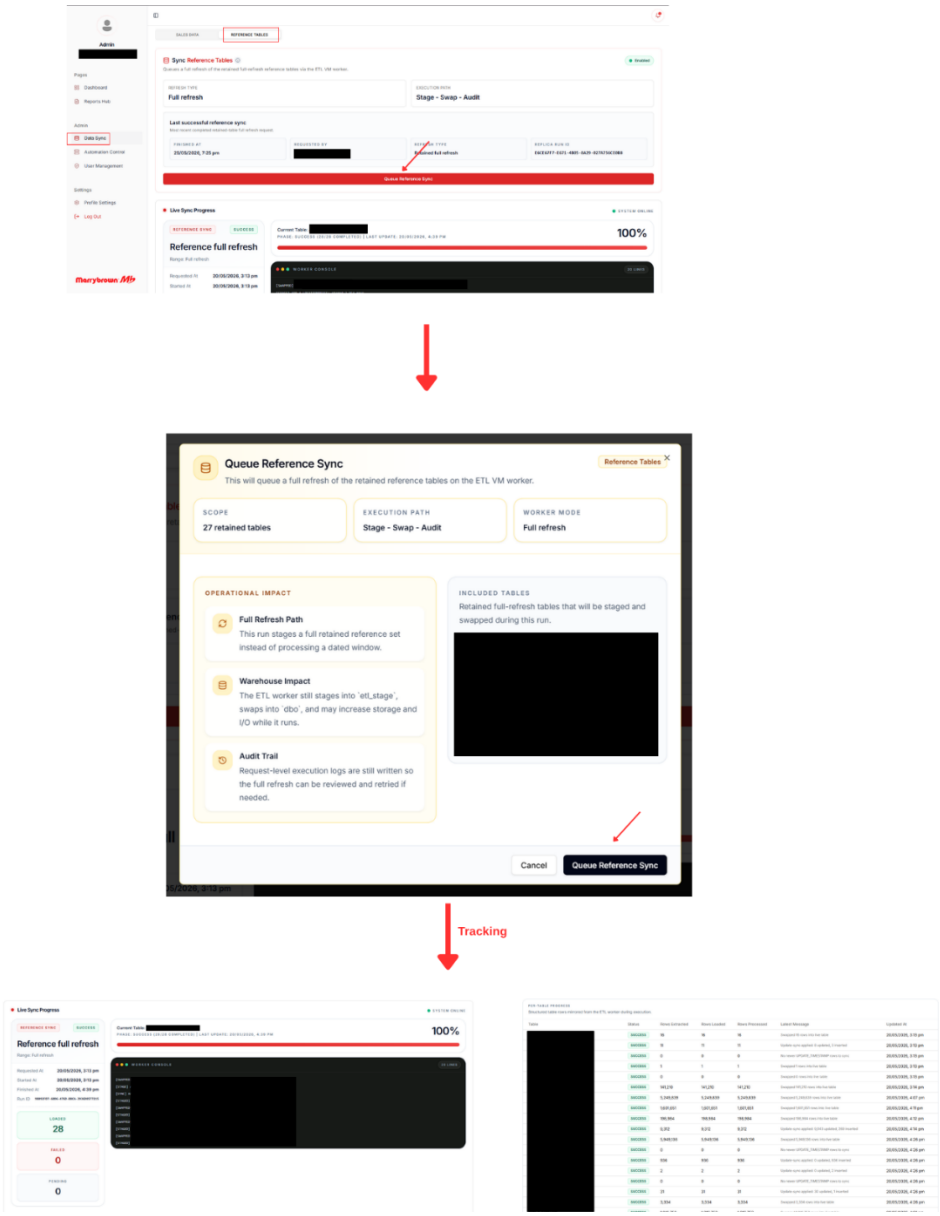

Figure 5.5: Portal-triggered reference-table refresh and execution tracking

Figure 5.6: Portal-triggered dated sales-data replication and data-quality review

5.3.2 Report Reconstruction Process

The main report-building process began with identifying the report behaviour expected from the vendor portal, including required parameters, visible output fields, grouping behaviour, totals, and export structure. From that baseline, the contributing datasets were traced in the replicated database and the required relationships were reconstructed at query level. This process was repeated across reports, even though the exact grouping and metric logic differed from one report family to another.

Once the contributing data paths were understood, the report logic was implemented in the FastAPI semantic/API layer. This typically included parameter validation, date-window scoping, outlet filtering, sales-status or payment-status handling where relevant, grouping and aggregation rules, and output formatting suitable for frontend rendering and export. The portal did not simply render raw SQL output. Instead, the backend response contracts were shaped to support tabular display, grouped subtotal behaviour, advanced search, and export actions in a stable and repeatable way.

Although each report had unique business rules, the implemented process remained consistent at high level. All report modules followed the same core pattern of source tracing, semantic reconstruction, parameter-driven querying, structured output shaping, and parity-oriented validation. This shared process allowed the project to scale from the retained formal report set to additional customised reports without abandoning the same platform architecture.

This reconstruction pattern is summarised in Figure 5.7. The figure represents the general backend process through which a report request moves from user parameter selection to semantic validation, replicated-data retrieval, dataset joining, grouping and derived-field handling, and final response shaping for portal display and export. Although the detailed query logic differs across report modules, the same high-level reconstruction sequence was retained throughout the implementation. This consistency was important because it allowed the platform to support multiple formal-scope and

customised reports within the same semantic/API architecture while preserving traceability for parity validation.

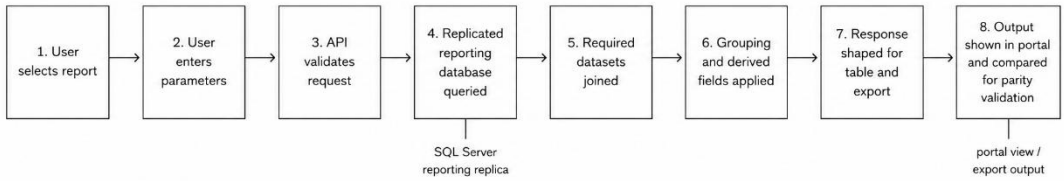

Figure 5.7: Sanitised illustration of report reconstruction logic

5.3.3 Selected Implementation Example: Payment Type (All Payment)

The Payment Type (All Payment) report is a suitable formal-scope example because it was operationally important, highly sensitive to row-level grouping behaviour, and representative of the reverse-engineering work required by the project. The purpose of this report was to provide a detailed breakdown of payment collections by store, order source, payment method, device, card type, and reference. The challenge was not merely to total payment rows, but to reconstruct the same grouping and display behaviour expected from the vendor portal with exactness sufficient for reconciliation work.

Implementation began by identifying the required payment-, sales-, and location-level source relationships in the replicated database. The main data sources were payment rows, sales status and order-source fields, and location-name mappings. Parameter handling was then defined around the selected date range, optional outlet scope, sales-status filtering, and payment-status behaviour. Because the report was intended for parity-sensitive operational use, the backend response also had to preserve a stable flat-row structure that could later support frontend grouped rendering and export processing.

The more difficult part of the implementation lay in the reference-grouping behaviour. Payment references could appear in multiple case variants, while portal output would still present them under a single preferred display value. Additional care was also required for payment-method categorisation, exclusion of return-sales behaviour, handling of zero-time anomalies, and consistent inclusion of payment rows in the correct collection totals. These issues made the report representative of the wider project challenge: the output could not be matched reliably through naive grouping alone, and several rounds of reverse engineering and refinement were required before parity could be accepted.

5.3.4 Selected Implementation Example: Voucher Campaign & Reward Sales Report

The Voucher Campaign & Reward Sales Report is a suitable customised-report example because it demonstrates that the implemented platform supported not only vendor-aligned report reconstruction, but also additional company-requested analysis built on the same architecture. This report was developed as a dynamic report family for eligible voucher campaigns and points rewards, while keeping the earlier Roblox-specific report intact as a separate implemented report.

The implementation process began with investigation of how voucher campaign and reward data could be represented reliably using the replicated data model. Rather than anchoring the report on a fixed voucher name, the implemented design used a selectable redemption-type identity so that the report could be applied to multiple campaign or reward families. The backend then reconstructed sale-level voucher scope, basket rows, derived bundle grouping, and additional-purchase indicators. This required not only data retrieval, but also interpretation of basket structure so that the final output could distinguish between the target bundle itself, other vouchers in the same sale, and additional paid items.

On the portal side, the customised report reused the shared report architecture but introduced a selector-driven reporting flow before ordinary date and outlet

filtering. This illustrates a key implementation result of the project: once the shared ETL, semantic/API, and portal architecture was stable, it became feasible to add company-requested customised reports without changing the overall platform model. The report therefore serves as evidence that the delivered system functioned as an extensible internal reporting platform rather than only as a one-time replication exercise.

5.3.5 Query Performance Observation

During implementation, selected report modules also underwent query-shape refinement and warehouse-side tuning to improve practical response time without changing validated business output. This work was documented through before-and-after timing checks on representative report scopes and was supported by later portal-side timing observations under real usage-style parameter sets.

One of the clearest examples was the Payment Type (All Payment) report, where the implementation logs recorded both application-side query-path refinement and later warehouse-side index recovery.

Table 5.1 summarises the strongest recorded timing observations for the Payment Type (All Payment) report.

Tested scope	**Before optimisation**	**After optimisation**	**Observation**
Single-store, single-day, 4-row case	approximately 63s	approximately 3.3s	substantial reduction after backend query-path refinement
Single-store, single-day, 608-row case	timed out beyond 120s	approximately 8.9s	timeout path removed after backend query-path refinement

Busy single-store, one-day sample after index recovery	not restated in the later log pass	approximately 1.1s	strong practical result after rebuilding disabled sales indexes
All-store, one-day sample after index recovery	not restated in the later log pass	approximately 12.2s	acceptable all-store one-day timing after warehouse-side tuning

Table 5.1: Recorded timing observations for Payment Type (All Payment)

The Payment Type observations indicate that performance improvement was achieved not only through report-query restructuring, but also through supporting warehouse maintenance and index recovery work.

Additional implementation logs recorded similar timing checks for other report families. Table 5.2 presents a compact summary of selected supporting observations.

Report	**Tested scope**	**Recorded timing change or practical timing**	**Interpretation**
Mobile Ordering Sales	All stores, 2025-01-01 to 2025-01-05	approximately 4.83s to 4.38s	modest improvement after query-shape refinement
Delivery-FoodPanda,Grabfood,ShopeeFood	All stores, 2025-01-01 to 2025-01-05	approximately 29.63s to 27.39s; later portal-side timing almost 20s	moderate backend improvement with usable portal-side outcome

Foodpanda Discount	All stores, 2025-01-01 to 2025-01-05	approximately 37.91s to 25.90s; later portal-side timing around 40+s	broad-window improvement, though still relatively heavy in portal use
Sales Return Report	All stores, 2025-01-01 to 2025-01-05	approximately 5.53s to 6.04s; later portal-side timing around 10s	timing was recorded and operationally acceptable, but not a strong improvement case
Sales Cancelled Report	All stores, 2025-01-01 to 2025-01-05	approximately 9.86s to 10.56s; later portal-side timing almost 20+s	timing was recorded and operationally acceptable, but not a strong improvement case
Deleted Items	All stores, 2025-01-01 to 2025-01-05	approximately 8.21s to 7.41s; later portal-side timing almost 8s	small but measurable improvement with acceptable practical timing

Table 5.2: Selected supporting timing observations across optimised report modules

Taken together, the recorded outcomes show that some report paths improved materially, while others improved more modestly or remained relatively heavy for broad all-store reporting windows. Practical portal-side timing checks were also

recorded after optimisation to ensure that the observed improvements remained meaningful from the user perspective rather than only at backend execution level.

These observations should be interpreted as implementation evidence of operational tuning rather than as a formal benchmark against the vendor portal. Within the project boundary, the main purpose of this work was to reduce timeout risk, improve response consistency, and keep validated report modules usable in the internal reporting workflow.

5.3.6 Report Accuracy Validation

The strongest testing evidence for the project was the report-accuracy validation process. This process was distinct from UAT because its purpose was not merely to confirm that a page loaded or exported correctly, but to confirm that the rebuilt report output was sufficiently aligned with the vendor portal under the same report conditions. The general validation approach was to run the platform report and the vendor portal report using the same date range, outlet scope, and other relevant filters, then compare row-level output, grouping behaviour, totals, subtotals, and exported results.

For the Payment Type (All Payment) report, this validation process was especially important because the report was sensitive to grouping precision at payment-reference level. The validation work therefore involved repeated comparison between platform output and vendor export output, investigation of mismatched rows or totals, and refinement of the reconstructed query logic until acceptable parity was achieved. Where anomalies remained, the investigation distinguished between logic faults and data-quality issues in the replica state, so that the report logic itself was not altered incorrectly to compensate for replica-specific data problems.

The same validation pattern was applied across the wider retained report surface. For each report module, parity validation used traceable filter conditions and visible output comparison rather than unsupported assumptions about hidden vendor

logic. This validation discipline was a defining characteristic of the project because it anchored report acceptance to evidence rather than to superficial interface completion. The detailed validation evidence is represented by the PV test cases shown in Figures 5.8 to 5.15.

PV01: Payment Type (All Payment) Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Payment Type (All Payment)	**Validation Type**	Report output reconciliation
Test Data Scope	Selected outlet and date range	**Validation Source**	Platform output compared with vendor portal output
Reviewer	Project developer and business user	**Execution Date**	2026-05-20

Validation Focus:
Validates payment-method grouping, outlet/date filters, transaction counts, totals, and export output.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Filter criteria	Same outlet and date range are applied	Platform accepted identical report parameters	Vendor portal export was generated using the same criteria	**Matched**	No filter mismatch identified
Payment grouping	Payment method categories are grouped consistently	Payment groups displayed under expected headings	Portal output used the same payment grouping pattern	**Matched**	Grouping names were checked manually
Transaction count	Number of payment records should match for the selected range	Record count aligned after refreshed replica data was used	Portal export showed the same transaction scope	**Matched**	Validated after sync completion
Total amount	Report-level total should reconcile	Displayed total matched the comparison value	Portal total matched the same total amount	**Matched**	Rounding reviewed at two-decimal level
Export output	Exported file should preserve displayed columns	Excel export contained the same visible report fields	Portal export was used as comparison evidence	**Matched**	Column order checked for report readability

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.8: PV01 Payment Type (All Payment) report accuracy validation

PV02: Sales Return Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Sales Return	**Validation Type**	Report output reconciliation
Test Data Scope	Return transactions within selected date range	**Validation Source**	Platform output compared with vendor portal output
Reviewer	Project developer and business user	**Execution Date**	2026-05-21

Validation Focus:
Validates return transaction retrieval, negative amount handling, return item details, and outlet/date filtering.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Date and outlet filters	Return report should only show selected business period and outlet	Returned rows followed selected parameter values	Portal output used equivalent period and outlet	**Matched**	Filter controls checked before comparison
Returned item details	Item lines should represent returned products only	Return item rows were displayed with expected quantities	Portal export contained the same return-line scope	**Matched**	Duplicate checking performed
Amount sign treatment	Return amounts should be represented consistently	Platform applied the expected return amount presentation	Portal output used the same direction of amount treatment	**Matched**	No manual sign conversion required
Subtotal and report total	Computed total should reconcile with portal result	Subtotal and grand total aligned	Portal total matched the same comparison amount	**Matched**	Rounding reviewed
Export result	Export should preserve selected report output	Export file generated successfully	Portal comparison export was available	**Matched**	No field loss observed

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.9: PV02 Sales Return Report accuracy validation

PV03: Sales Cancelled Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Sales Cancelled	**Validation Type**	Report output reconciliation
Test Data Scope	Cancelled transactions in selected business period	**Validation Source**	Platform output compared with vendor portal output
Reviewer	Project developer and business user	**Execution Date**	2026-05-21

Validation Focus:
Validates cancelled order inclusion rules, cancellation-related columns, totals, and export behaviour.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Cancellation scope	Only cancelled sales should appear	Rows represented cancelled transactions only	Portal comparison output used the same cancelled-sales scope	**Matched**	Non-cancelled transactions excluded
Order identifiers	Cancelled transaction references should align	Displayed transaction identifiers matched sampled portal records	Portal export showed same sampled identifiers	**Matched**	Sampling performed across outlets
Cancellation details	Reason/status fields should remain traceable where available	Cancellation-related fields displayed in expected columns	Portal output contained equivalent cancellation references	**Matched**	Missing optional values treated as blank
Total amount	Cancelled sales amount should reconcile	Displayed total matched portal comparison total	Portal output total used for reconciliation	**Matched**	Rounding checked
Export result	Export should preserve cancelled sales rows and columns	Export generated and retained visible fields	Portal output used as comparison record	**Matched**	Export column visibility checked

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.10: PV03 Sales Cancelled Report accuracy validation

PV04: Sale Delivery Ex Tax Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Sale Delivery Ex Tax	**Validation Type**	Report output reconciliation
Test Data Scope	Delivery transactions for selected outlet/date range	**Validation Source**	Platform output compared with vendor portal output
Reviewer	Project developer and business user	**Execution Date**	2026-05-22

Validation Focus:
Validates delivery sales calculation excluding tax, delivery-channel grouping, and date/outlet filtering.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Delivery transaction scope	Only delivery-related transactions should be included	Displayed rows followed delivery scope	Portal output used same delivery report context	**Matched**	Channel filtering reviewed
Ex-tax amount calculation	Sales amount should exclude tax component	Platform amount matched ex-tax comparison value	Portal result used as ex-tax reference	**Matched**	Calculation checked at sampled row and total level
Grouping by outlet/date	Rows should group consistently by selected outlet and date	Output grouped using expected outlet/date fields	Portal export followed equivalent grouping	**Matched**	Group headers reviewed
Report total	Final total should reconcile	Displayed total matched comparison total	Portal export showed same report total	**Matched**	No unexplained difference after data refresh
Export result	Export should preserve ex-tax values	Excel export preserved calculation columns	Portal export used as comparison evidence	**Matched**	No formatting issue affecting totals

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.11: PV04 Sale Delivery Ex Tax report accuracy validation

PV05: MB Cash Voucher Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	MB Cash Voucher	**Validation Type**	Report output reconciliation
Test Data Scope	Voucher redemption activity in selected period	**Validation Source**	Platform output compared with vendor portal output
Reviewer	Project developer and business user	**Execution Date**	2026-05-22

Validation Focus:
Validates voucher redemption rows, voucher amounts, outlet/date filters, and grouped totals.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Voucher redemption scope	Only relevant voucher redemption rows should appear	Voucher rows were filtered to the selected redemption scope	Portal export reflected the same voucher activity	**Matched**	Non-voucher payment rows excluded
Voucher amount	Voucher amount should match row-level comparison	Sampled voucher amounts aligned	Portal sampled rows showed same amounts	**Matched**	Sampled across more than one outlet
Outlet/date grouping	Report grouping should follow selected outlet/date parameters	Output followed expected grouping	Portal output grouped in the same business context	**Matched**	Group totals checked
Grand total	Total voucher value should reconcile	Platform total aligned with comparison total	Portal output showed same total value	**Matched**	Rounding reviewed
Export result	Export should preserve voucher identifiers and values	Export file generated with visible voucher fields	Portal export used for comparison	**Matched**	No field omission observed

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.12: PV05 MB Cash Voucher redemption accuracy validation

PV06: Voucher Campaign & Reward Sales Report Accuracy Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Voucher Campaign & Reward Sales	**Validation Type**	Custom report validation
Test Data Scope	Selected campaign period and reward/voucher criteria	**Validation Source**	Platform output compared with business-requested rules and portal evidence
Reviewer	Project developer and business user	**Execution Date**	2026-05-24

Validation Focus:
Validates customised voucher campaign filtering, reward sales grouping, and campaign-level totals.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Campaign filter	Only requested campaign/reward rows should be included	Platform filtered rows using the configured campaign criteria	Comparison evidence confirmed expected campaign scope	**Matched**	Custom request reviewed with business user
Voucher/reward identification	Voucher and reward records should be classified correctly	Rows were separated into the expected campaign/reward categories	Reference output supported same categorisation	**Matched**	Category samples checked
Outlet and date filters	Selected outlet/date criteria should limit the output	Output respected selected criteria	Reference output used same period and outlet context	**Matched**	Parameter handling verified
Campaign total	Total campaign-related value should reconcile	Campaign total matched validation total	Comparison evidence matched the same total	**Matched**	Rounding reviewed
Export result	Export should support operational review	Export contained campaign fields and values	Reference export used for validation	**Matched**	No sensitive setup data included

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
None identified	N/A	No adjustment required after comparison	Closed

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.13: PV06 Voucher Campaign and Reward Sales accuracy validation

PV07: Portal-Triggered Data Quality Validation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Portal-triggered replication quality check	**Validation Type**	Data completeness and consistency validation
Test Data Scope	Selected reporting period after manual sync	**Validation Source**	Replica status, quality-check result, and report output comparison
Reviewer	Project developer and business user	**Execution Date**	2026-05-25

Validation Focus:
Validates that replicated reporting data can be checked and rechecked after a portal-triggered sync request.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	**Passed**	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Sync request completion	Manual sync should finish before validation is performed	Sync status reached completed state	Operational status view confirmed completion	**Matched**	No direct server credentials documented
Row-count reasonableness	Replica row counts should be consistent for selected period	Quality check returned expected completeness result	Operational comparison did not show missing period after refresh	**Matched**	Focused on reporting readiness
Missing-period detection	System should identify incomplete data when present	Quality check exposed incomplete state before repair/recheck where applicable	Expected behaviour observed	**Matched**	Scenario tested using controlled period selection
Recheck after repair/sync	Recheck should update status after data refresh	Recheck result changed to valid after refreshed data was available	Operational status reflected updated state	**Matched**	Closed through supported portal path
Report readiness	Reports should be generated only against refreshed replica data	Report validation used refreshed replica period	Comparison output used equivalent business period	**Matched**	Supports final report accuracy validation

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
Incomplete historical periods	Operational data completeness issue	Handled through manual sync and quality recheck	Closed for validated periods

Validation Conclusion

The reconstructed output was accepted for the validated scenario because the selected rows, grouping, totals, and export behaviour reconciled with the comparison evidence.

Figure 5.14: PV07 Portal-triggered data quality validation

PV08: Product Mix and Discount Profit Logic Validation Limitation

Report accuracy validation evidence. This layout is comparison-based and is intentionally different from the UAT test case template.

Report / Process	Product Mix / Discount Remark profit-related output	Validation Type	Exception and limitation record
Test Data Scope	Selected report samples with profit-related fields	Validation Source	Platform reconstruction compared with vendor portal output
Reviewer	Project developer and business user	Execution Date	2026-05-26

Validation Focus:
Documents the unresolved profit-logic limitation for Product Mix related reporting and Discount Remark reporting.

Comparison Setup

Platform Output Evidence	Vendor / Reference Output Evidence	Filter Alignment	Comparison Level	Validation Method	Reviewer Note	Result	Evidence Location	Sensitivity Check	Public-Safe Note
Portal/API generated output and export file	Vendor portal output or approved business comparison output	Same selected outlet/date/report criteria	Row, group, subtotal, total, and export level	Manual reconciliation supported by sampled row checks	Reviewed by developer and business user	Not Fully Closed	Screenshot/export kept separately for final report evidence	Credentials and internal access paths excluded	Only report behaviour and validation outcome documented

Reconciliation Checklist

Comparison Point	Expected Match / Rule	Platform Observation	Vendor / Reference Observation	Result	Notes
Basic row scope	Product/discount rows should follow selected date and outlet filters	Row scope could be reconstructed for selected samples	Portal output provided comparable row scope	Matched	Base filtering was not the blocker
Quantity and sales values	Non-profit values should align where source logic was traceable	Quantity and sales-related fields were validated in sampled output	Portal output supported these comparisons	Matched	Validation focused on traceable fields
Profit column logic	Profit-related value should reconcile with portal result	Remaining profit calculation could not be isolated completely	Portal output did not provide enough observable evidence to derive final logic	Not Fully Closed	Documented as external black-box logic limitation
Report closure decision	Only fully validated reports should be included in retained active surface	Report remained documented but not treated as fully closed	Comparison limitation remained unresolved	Not Fully Closed	Avoided claiming false parity
Academic reporting treatment	Limitation should be stated honestly	Limitation recorded in final-report evidence	N/A	Documented	No internal credentials or proprietary query text included

Variance / Exception Handling

Data Item / Field	Variance Type	Investigation / Handling Action	Resolution Status
Profit column	Unresolved parity difference	Reverse engineering and validation were not sufficient to isolate final profit logic	Not fully closed

Validation Conclusion

The report area was documented as not fully closed because the remaining profit-related logic could not be isolated with sufficient confidence through reverse engineering and portal comparison evidence.

Figure 5.15: PV08 Product Mix and discount profit logic validation boundary

5.3.7 Black-Box Testing and User Acceptance Testing

Black-box testing and User Acceptance Testing (UAT) were treated together because, in the context of this project, both evaluated the system from the user-facing perspective rather than from the internal code path. These tests focused on whether users could sign in, access reports, enter parameters, retrieve outputs, inspect totals, export data, and use role-restricted administrative functions appropriately. The intent was to verify that the implemented platform supported realistic operational workflows rather than merely exposing technically correct endpoints.

The test scenarios were structured as test cases using the TC prefix, such as TC01, TC02, and subsequent cases. Core user-facing scenarios included

authentication, report-catalogue access, report retrieval with valid filters, export behaviour, access to customised report modules, manual sync submission, data-quality follow-up visibility, automation-control access, and role-restricted administrative behaviour. This structure provided a clear and traceable record of functional acceptance across the implemented user workflows.

The UAT and black-box testing outcomes were intended to show that the implemented platform could be used coherently by both ordinary reporting users and authorised administrative users. This was especially relevant for the data-sync and automation functions, because these features extended the platform beyond simple report viewing and into controlled reporting operations. By structuring these tests around user-observable behaviour, the testing section complements the report-accuracy validation process rather than duplicating it.

The detailed UAT and black-box test cases are shown in Figures 5.16 to 5.31.

User Login with Valid Credentials		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that an authorised user can log in and reach the protected portal area.	Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC01	Verifies that an authorised user can log in and reach the protected portal area.	Registered active user account; login page accessible.	User email and password provided by test administrator.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Navigate to the login page.
2. Enter a valid email address.
3. Enter a valid password.
4. Click the login button.
5. Observe the landing page after authentication.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The user is redirected to the reporting portal after successful authentication.

Actual Outcome

The user logged in successfully and reached the protected portal page.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.16: TC01 user login with valid credentials

Reports Hub Access			
Description:	Verifies that an authenticated user can open the reports hub and view available report categories.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC02	Verifies that an authenticated user can open the reports hub and view available report categories.	User is logged in; report definitions are available.	Authenticated user session.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open the portal navigation menu.
2. Select Reports.
3. Review displayed report categories.
4. Select one report category.
5. Confirm that the report page opens.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The reports hub displays available report options and allows navigation to a report page.

Actual Outcome

Reports hub loaded successfully and report navigation worked as expected.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.17: TC02 Reports Hub access

Report Parameter Entry			
Description:	Verifies that a user can enter common report filters such as date range and outlet before generating a report.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Functional / Black-box

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC03	Verifies that a user can enter common report filters such as date range and outlet before generating a report.	User has access to a report page.	Outlet/date range selected for testing.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open a report page.
2. Select a valid start date.
3. Select a valid end date.
4. Select or confirm outlet/report filters.
5. Submit the report request.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The system accepts valid report parameters and starts report generation.

Actual Outcome

Parameters were accepted and the report request was processed.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.18: TC03 report parameter loading

Generate Payment Type Report			
Description:	Verifies that a user can generate the Payment Type (All Payment) report from the portal.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC04	Verifies that a user can generate the Payment Type (All Payment) report from the portal.	User is logged in and report data is available for selected period.	Payment Type report with selected outlet/date range.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open Payment Type report.
2. Enter required filter values.
3. Click Search or Generate.
4. Wait for table output.
5. Review the loaded result rows.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The Payment Type report loads with grouped payment records and totals.

Actual Outcome

The report loaded with expected grouped rows and totals.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.19: TC04 Payment Type report query

Review Report Totals and Grouping			
Description:	Verifies that report rows, group totals, and grand totals are visible and understandable to the user.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	Medium
		Test Type:	Functional / Black-box

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC05	Verifies that report rows, group totals, and grand totals are visible and understandable to the user.	Report output has been generated.	Generated report result table.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Generate a report with valid filters.
2. Scroll through grouped report output.
3. Review subtotal rows where available.
4. Review grand total row.
5. Compare displayed values with expected validation evidence.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The report table presents grouped records and totals clearly.

Actual Outcome

Grouping and totals were displayed clearly for review.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.20: TC05 report totals review

Export Report Output			
Description:	Verifies that a user can export generated report data for operational use.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC06	Verifies that a user can export generated report data for operational use.	Report output is already loaded on screen.	Generated report table.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Generate a report.
2. Click the export button.
3. Wait for file generation.
4. Open the exported file.
5. Compare exported columns with visible report output.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The exported file contains the displayed report fields and values.

Actual Outcome

Export completed successfully and contained the expected report output.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.21: TC06 report output export

Advanced Search and Table Filtering			
Description:	Verifies that the user can narrow report output using available table search or filtering controls.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	Medium
		Test Type:	Functional / Black-box

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC07	Verifies that the user can narrow report output using available table search or filtering controls.	Report output contains multiple rows.	Search keyword or filter value.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Generate a report with multiple rows.
2. Enter a search keyword or filter value.
3. Apply the search/filter control.
4. Review filtered rows.
5. Clear the filter and confirm full result returns.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The table filters results according to the entered criteria and restores full output when cleared.

Actual Outcome

Filtering behaved as expected and the result table was restored after clearing.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.22: TC07 advanced search behaviour

Generate Sales Return Report			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that a user can retrieve a sales-return related report from the portal.	Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC08	Verifies that a user can retrieve a sales-return related report from the portal.	User is logged in and return records exist for the selected period.	Sales Return report filters.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open Sales Return report.
2. Enter selected date and outlet filters.
3. Generate the report.
4. Review return transaction rows.
5. Review totals and export availability.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The Sales Return report displays return transaction rows for the selected filter.

Actual Outcome

Sales Return report loaded correctly with expected rows and totals.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.23: TC08 Sales Return Report query

Generate Custom Voucher Campaign Report			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that a customised report requested by the company can be generated through the portal.	Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC09	Verifies that a customised report requested by the company can be generated through the portal.	User is logged in; customised report is enabled.	Voucher campaign date/filter criteria.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open the Voucher Campaign & Reward Sales report.
2. Enter campaign-related filter values.
3. Generate the report.
4. Review campaign rows and totals.
5. Export the result if required.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The customised report displays campaign-related sales/reward output according to selected criteria.

Actual Outcome

The customised report generated successfully and output was reviewable.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.24: TC09 customised report query

Admin Page Access Restriction			
Description:	Verifies that administrative pages are restricted from normal report users.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Security / Black-box

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC10	Verifies that administrative pages are restricted from normal report users.	One normal user and one admin user are available for testing.	Normal user session and admin user session.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Log in as a normal user.
2. Attempt to access an admin-only page.
3. Observe access behaviour.
4. Log out and log in as admin user.
5. Open the same admin page.
6.
7.
8.

Postconditions

Access remains controlled based on role.

Execution Date

2026-05-28

Expected Outcome

Normal user is blocked from admin functions, while admin user can access them.

Actual Outcome

Normal user could not access admin-only page and admin user could access it.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.25: TC10 normal-user admin restriction

Open Data Sync Administration Page			
Description:	Verifies that an admin user can open the data sync page and view sync-related controls/status.	Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
		Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC11	Verifies that an admin user can open the data sync page and view sync-related controls/status.	Admin user is logged in.	Admin account and sync page route.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Log in as an admin user.
2. Open the Data Sync page.
3. Review available sync controls.
4. Review latest status area.
5. Confirm no unauthorised credentials are displayed.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The data sync page opens for admin users and shows appropriate controls/status information.

Actual Outcome

Data Sync page opened and displayed operational controls/status safely.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.26: TC11 admin Data Sync access

Submit Portal-Triggered Manual Sync			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that an admin user can submit a manual sync request through the portal.	Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC12	Verifies that an admin user can submit a manual sync request through the portal.	Admin user is logged in; sync service is available.	Selected sync period or sync option.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open Data Sync page.
2. Select the required sync option.
3. Submit manual sync request.
4. Confirm the request prompt if displayed.
5. Review initial sync status.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The system accepts the sync request and records it for processing.

Actual Outcome

Manual sync request was submitted and appeared in the sync status view.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.27: TC12 manual sync request submission

Monitor Sync Progress			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that an admin user can monitor portal-triggered sync progress after submission.	Priority:	Medium
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC13	Verifies that an admin user can monitor portal-triggered sync progress after submission.	A manual sync request has been submitted.	Existing sync request record.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open the sync status section.
2. Refresh or wait for status update.
3. Review current status.
4. Confirm completion or failure message.
5. Open related report after sync if needed.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The system displays an understandable sync status and completion result.

Actual Outcome

Sync progress/status was visible and understandable to the admin user.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.28: TC13 sync progress review

Run Data Quality Recheck			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that an admin user can trigger or review a data quality recheck after sync.	Priority:	High
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC14	Verifies that an admin user can trigger or review a data quality recheck after sync.	Admin user is logged in; sync has completed.	Selected reporting period for quality check.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open Data Sync or quality-check area.
2. Select the reporting period.
3. Run or refresh the quality check.
4. Review completeness/status result.
5. Use result to decide whether reports are ready.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The system displays data quality status for the selected period.

Actual Outcome

Data quality recheck completed and showed an understandable status result.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.29: TC14 data-quality re-check workflow

Automation Control Management			
		Author:	Project Developer
		Reviewer:	Project Supervisor / Company User
Description:	Verifies that an admin user can review and manage ETL automation controls from the portal.	Priority:	Medium
		Test Type:	Functional / UAT

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC15	Verifies that an admin user can review and manage ETL automation controls from the portal.	Admin user is logged in; automation control page is enabled.	Automation control page and selected schedule/control option.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open Automation Control page.
2. Review current automation status.
3. Select an available management action.
4. Submit the action.
5. Confirm updated status/message.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The automation page displays current status and records the selected management action.

Actual Outcome

Automation status and control action were displayed clearly.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.30: TC15 Automation Control review

User Management Workflow		
Description:	Verifies that an admin user can manage portal users according to assigned role and status.	

Author:	Project Developer	
Reviewer:	Project Supervisor / Company User	
Priority:	Medium	
Test Type:	Functional / UAT	

Test Case ID	Objective	Preconditions	Input Data	Test Environment	Data Preparation
TC16	Verifies that an admin user can manage portal users according to assigned role and status.	Admin user is logged in.	Test user account data without real credentials.	Windows browser environment and deployed reporting portal	Use non-sensitive test account and selected report period prepared for validation

Test Steps

1. Open User Management page.
2. Review user list.
3. Open create/edit user form.
4. Enter non-sensitive test user details.
5. Save and review the updated user list.
6.
7.
8.

Postconditions

System state remains available for continued testing or user work.

Execution Date

2026-05-28

Expected Outcome

The user management page allows admin users to maintain user information safely.

Actual Outcome

User management workflow was completed using non-sensitive test data.

Bugs Encountered

1. No issues encountered.
2.
3.

Figure 5.31: TC16 User Management workflow

5.3.8 White-Box Testing

White-box testing was used to evaluate selected internal processing logic that could not be assessed fully through user-facing behaviour alone. In this project, white-box testing did not aim to provide exhaustive code-coverage measurement. Instead, it focused on the internal logic that had direct impact on report trustworthiness and operational safety, such as request-parameter validation, boundary-based refresh handling, source-to-replica quality-check flow, grouping and aggregation logic in report reconstruction, export-structure consistency, and role-restricted access handling.

This form of testing was relevant because a report can appear correct from the interface while still containing internal weaknesses in parameter interpretation, refresh-window replacement logic, or grouped output construction. By checking selected internal logic paths explicitly, the implementation could be reviewed not only at output level but also at process level. In academic terms, this strengthened the credibility of the completed system because it demonstrated that testing attention was given to both visible behaviour and the underlying logic that produced that behaviour. The detailed white-box test cases are shown in Figures 5.32 to 5.40.

WB01: API Parameter Validation Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB01	**Module / Component**	Backend API router and request schema
Test Method	Boundary and invalid-input checks	**Risk Area**	Checks that backend report endpoints validate required filters before executing report logic.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that backend report endpoints validate required filters before executing report logic.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Required date parameter	Missing date should be rejected before query execution	Validation layer returned controlled error response	Passed
2	Date range order	Invalid start/end sequence should not produce misleading output	Invalid period was rejected or normalised by validation path	Passed
3	Outlet/report filter	Unexpected filter value should not break report execution	Controlled validation behaviour observed	Passed
4	Response consistency	Valid request should return structured report payload	Successful response followed expected JSON structure	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.32: WB01 API parameter validation

WB02: Payment Grouping and Total Calculation Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB02	**Module / Component**	Payment Type report service/query layer
Test Method	Grouping and aggregation path review	**Risk Area**	Checks internal grouping and total logic for payment-oriented report reconstruction.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks internal grouping and total logic for payment-oriented report reconstruction.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Grouping key	Payment rows should be grouped by the intended payment classification	Grouped output followed expected payment categories	Passed
2	Aggregation	Amounts should be summed at group and report-total level	Subtotal and total calculations aligned with validation evidence	Passed
3	Null/blank handling	Blank optional fields should not break grouping	Rows with optional blanks remained processable	Passed
4	Export mapping	Exported totals should match displayed totals	Displayed and exported totals aligned	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.33: WB02 Payment Type grouping logic

WB03: ETL Re-run and Idempotency Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB03	**Module / Component**	ETL scripts and reporting replica load flow
Test Method	Controlled rerun/checkpoint review	**Risk Area**	Checks that repeated ETL execution for the same reporting period does not create duplicated reporting output.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that repeated ETL execution for the same reporting period does not create duplicated reporting output.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	**Passed**	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Existing period load	Rerun should refresh or preserve target period without duplicate rows	Repeated run did not create duplicate report rows in validated period	**Passed**
2	Load boundary	ETL should apply selected period boundary consistently	Loaded records stayed within selected sync boundary	**Passed**
3	Failure handling	Failed step should not be treated as successful completion	Status remained traceable for failed/incomplete cases	**Passed**
4	Post-load reporting	Reports should read from refreshed replica state	Report output reflected refreshed period after completion	**Passed**

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	**Passed**	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	**Passed**	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	**Passed**	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.34: WB03 boundary-based ETL rerun

WB04: Staged Refresh Window Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB04	**Module / Component**	Replica refresh and backfill process
Test Method	Refresh-boundary logic review	**Risk Area**	Checks that controlled refresh/backfill windows support historical completeness without uncontrolled production changes.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that controlled refresh/backfill windows support historical completeness without uncontrolled production changes.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	**Passed**	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Selected period	Refresh should target selected historical/current period	Refresh scope followed selected period boundary	**Passed**
2	Historical gaps	Known incomplete periods should remain identifiable	Quality check highlighted incomplete state where applicable	**Passed**
3	Operational control	Unattended changes should be managed through approved controls	Automation controls and manual sync path were separated clearly	**Passed**
4	Report readiness	Report validation should be performed after refresh	Validation was performed after refreshed data was available	**Passed**

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	**Passed**	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	**Passed**	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	**Passed**	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.35: WB04 staged refresh safety

WB05: Data Quality Detection Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB05	**Module / Component**	Data quality check process
Test Method	Completeness and consistency checkpoints	**Risk Area**	Checks that the platform can identify data completeness issues after replication.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that the platform can identify data completeness issues after replication.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes				
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented				

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Completeness check	System should detect incomplete reporting period	Incomplete state was visible through quality-check result	Passed
2	Status display	Quality status should be understandable to admin user	Status was displayed in admin workflow	Passed
3	No credential exposure	Quality result should not expose server credentials	Only safe status information was shown	Passed
4	Follow-up action	Admin should know whether recheck/sync is needed	Result supported operational decision	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note					
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded					
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules					
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety					

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.36: WB05 source-to-replica data quality check

WB06: Data Quality Repair and Recheck Routing

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB06	**Module / Component**	Manual sync worker and data quality recheck flow
Test Method	Workflow checkpoint review	**Risk Area**	Checks the internal process flow from detected issue to repair/sync and recheck.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks the internal process flow from detected issue to repair/sync and recheck.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Issue detection	Incomplete period should trigger recheck or sync action	Detected issue was routed to supported sync/recheck flow	Passed
2	Repair/sync action	Manual sync should be able to refresh selected data scope	Sync request processed through portal-supported path	Passed
3	Recheck	Quality status should update after data refresh	Recheck reflected refreshed data state	Passed
4	Report reuse	Reports should use updated replica data after recheck	Report validation used refreshed output	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.37: WB06 quality repair request flow

WB07: Report Export Mapping Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB07	**Module / Component**	Frontend report export utility and report table model
Test Method	Display-to-export mapping review	**Risk Area**	Checks that displayed report data and exported report data remain consistent.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that displayed report data and exported report data remain consistent.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Visible columns	Export should include expected visible fields	Exported file retained report columns needed for review	Passed
2	Value consistency	Exported values should match table display	Exported values matched displayed report rows in sample checks	Passed
3	Totals	Export should not alter report totals	Exported totals aligned with on-screen totals where included	Passed
4	Formatting safety	Export should not include hidden credentials or internal paths	Export contained report data only	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.38: WB07 report export mapping

WB08: Role-Based Access Guard Logic

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB08	**Module / Component**	Authentication/authorization layer and portal routing
Test Method	Access-control path review	**Risk Area**	Checks that route and backend access controls protect administrative features.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that route and backend access controls protect administrative features.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Normal user access	Normal user should not open admin pages	Admin-only routes were blocked for normal user scenario	Passed
2	Admin user access	Admin user should be able to open admin pages	Admin scenario could open admin pages	Passed
3	Backend protection	Protected operations should require authorized request	Admin operations followed protected access path	Passed
4	Navigation visibility	UI should not encourage unauthorised use	Admin navigation followed role-aware behaviour	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.39: WB08 role-based access restriction

WB09: Automation-Control Request Handling

White-box testing evidence. This layout focuses on internal logic checkpoints instead of user-facing UAT steps.

White-Box Test ID	WB09	**Module / Component**	Automation control API and admin frontend page
Test Method	Request/status checkpoint review	**Risk Area**	Checks that automation management requests are handled through controlled admin workflow.
Reviewer	Project developer	**Execution Date**	2026-05-29

Internal Logic Under Test:
Checks that automation management requests are handled through controlled admin workflow.

This sheet records internal checkpoints, controlled conditions, and observed processing behaviour. It is not a user acceptance test case.

Controlled Inputs and Preconditions

Input / Condition	Expected Internal Handling	Observed Handling	Evidence Method	Result	Notes
Validated report period, role, or request payload depending on module	Processing should follow defined validation, grouping, refresh, or access-control path	Observed behaviour followed expected internal route during test scenario	Code-path review, API response inspection, and report-output cross-check	Passed	No credentials, server addresses, or proprietary setup details documented

Internal Checkpoint Trace

No.	Internal Checkpoint	Expected Internal Behaviour	Observed Evidence	Result
1	Status retrieval	Admin page should retrieve current automation status	Status was available for display	Passed
2	Action submission	Supported action should be sent as controlled request	Action request followed admin workflow	Passed
3	Response handling	User should receive understandable result	Success/status message shown after action	Passed
4	Safety boundary	Automation controls should not expose implementation secrets	Only safe operational status/control information was displayed	Passed

Boundary / Negative Case Review

Boundary Case	Expected Protection	Observed Result	Status	Reviewer Note
Invalid or incomplete input	System should reject or handle safely without misleading report output	Controlled handling observed	Passed	No uncontrolled failure recorded
Unauthorised or inappropriate operation where relevant	System should restrict protected actions	Role/access guard remained effective in tested path	Passed	Applies to admin-related modules
Empty or incomplete data state	System should display understandable status rather than false result	Status/empty-state handling remained understandable	Passed	Supports operational safety

White-Box Test Conclusion

The selected internal logic path behaved as expected for the tested conditions. The test supports the implementation explanation in Chapter 5 without exposing internal credentials, server access instructions, or sensitive operational details.

Figure 5.40: WB09 automation-control request handling

5.4 Summary

This chapter documented the implementation and testing of the Marrybrown Sales and Payment Analytics Platform. It described the completed development of the reporting-database and ETL layer, the FastAPI semantic/API layer, the React portal layer, and the supporting administrative functions that made the platform operationally usable. It also outlined the formal report implementation coverage, the selected implementation examples, the recorded query-performance observation work, and the

testing approach used to evaluate report accuracy, user-facing behaviour, and selected internal logic.

Taken together, the implementation and testing outcomes show that the project moved beyond conceptual design and delivered a real internal reporting platform with validated report modules, controlled refresh capability, role-based access, and company-requested extensions. The next chapter discusses the platform's contribution, remaining constraints, and future suggestions.

Chapter 6

CONCLUSION

6.1 Introduction

This chapter concludes the final report for the Marrybrown Sales and Payment Analytics Platform. Whereas Chapter 5 focused on the implemented development and testing work, this chapter interprets the overall project outcome through the platform's contribution, achievement, remaining constraints, and future suggestions. The discussion therefore moves from implementation detail to final evaluation of what the project delivered within its defined academic and business boundary.

The chapter does not repeat the detailed report-construction, ETL, or testing procedures already documented earlier. Instead, it summarises the practical significance of the completed platform, explains the main limitations that remained at project closure, and identifies reasonable directions for future improvement if the platform is extended further in the company environment.

6.2 System Contribution/Achievement

The project successfully delivered the Marrybrown Sales and Payment Analytics Platform as a real internal reporting system deployed in a company environment rather than as a purely academic prototype. Its first major contribution was the establishment of a company-controlled reporting architecture that reduced operational dependence on the vendor portal for selected sales and payment reporting needs. By replicating selected source-system data into a Microsoft SQL Server reporting database, reconstructing report logic in a FastAPI semantic/API layer, and providing authenticated access through a React-based portal, the project created a

practical internal reporting path that could be managed and extended within Marrybrown's own operating context.

The second major contribution was report reconstruction with evidence-based validation. The project did not treat report development as simple page creation. Instead, it involved tracing source data, rebuilding vendor-aligned logic, testing parameter behaviour, and validating outputs against vendor-visible results. This gave the delivered platform traceable reporting behaviour for the retained report surface and demonstrated that operational reporting continuity can be achieved through disciplined reverse engineering and parity validation rather than through unsupported approximation.

The third contribution was operational usability. The implemented system provided more than report retrieval alone. Internal users were supported through report search, filtering, tabular review, grouped output, and export functions, while authorised administrative users were supported through user management, portal-triggered sync, automation-management visibility, and data-quality checking workflows. These features made the platform usable as an internal reporting environment rather than as a set of isolated backend endpoints or static output pages.

An additional achievement was architectural extensibility. Beyond the formal implementation scope, the project also demonstrated that the shared reporting architecture could support further company-requested report families without requiring a separate system model. This indicates that the platform functioned not only as a continuity solution for existing report needs, but also as a reusable internal base for future reporting extensions.

Taken together, these achievements show that the project delivered a validated internal reporting platform with practical business value. It addressed the company's reporting continuity problem, produced a company-usable deployed system, and documented a repeatable approach for report reconstruction, operational control, and controlled extension.

6.3 System Constraint

Despite the achieved outcome, several important constraints remained at project closure. The first constraint was that report parity depended not only on application logic, but also on the completeness and consistency of the replicated reporting data. In practice, some mismatches arose from historical incompleteness, late source updates, or replica-state conditions rather than from backend logic alone. This meant that accurate report delivery required both valid query logic and trustworthy underlying replicated data.

The second constraint was the black-box nature of the vendor reporting behaviour. The project reconstructed report outputs through observable behaviour, available datasets, and iterative validation, but the full underlying logic of the vendor portal was not directly exposed for all report areas. As a result, reverse engineering was necessarily evidence-led and iterative rather than fully specification-driven. This limitation increased the time required for validation and made some report areas more difficult to conclude with certainty.

The third constraint was that not every formal-scope report area was fully closed within the project boundary. Product Mix Report, Discount Remark Report, and Product Mix with modifier without ETL remained unresolved after reverse engineering and parity validation because the remaining profit-related logic could not be isolated conclusively. These unresolved areas should therefore be understood as genuine logic-boundary constraints rather than as simple page-development omissions.

Another constraint concerned performance variation across report families. Query tuning and supporting warehouse maintenance improved the practical response time of several reports, and the system reached an operationally usable level for the retained report surface. However, some broader all-store report windows still remained relatively heavy compared with narrower scopes. This means that, although the platform became usable and materially improved in several paths, performance

optimisation remained dependent on report shape, replica condition, and warehouse-side factors.

A final constraint is that the delivered platform should not be interpreted as a complete functional replacement for every capability of the vendor portal. The project was scoped around selected sales and payment reporting requirements requested by the company. Accordingly, the platform should be understood as a validated internal reporting solution for the agreed report surface, not as an exhaustive reimplementation of the external vendor ecosystem.

6.4 Future Suggestion

Several future suggestions follow naturally from the completed implementation. The first is continued investigation of unresolved profit-related report logic, especially in the product-mix-related report areas. If additional observable business evidence, validated reference behaviour, or clearer supporting data becomes available, these report modules may be revisited for further reconstruction and final parity validation.

The second suggestion is continued improvement of historical completeness and replicated-data reliability. Since report trustworthiness depends partly on source-to-replica consistency, future work may focus on stronger historical validation coverage, more efficient repair handling for incomplete windows, and continued operational refinement of sync and data-quality workflows.

The third suggestion is selective performance refinement for heavier report scopes. Although the implemented platform reached an operationally acceptable level for current use, some broad all-store report windows remained comparatively heavy. Future improvement may therefore focus on deeper warehouse-side tuning, query-plan refinement, and scope-specific optimisation where repeated operational demand justifies further effort.

The fourth suggestion is controlled expansion of validated report coverage. The established architecture already demonstrated that additional company-requested reports could be incorporated into the same platform model. Future work may therefore extend the platform by adding new report modules, provided that the same discipline of source tracing, logic reconstruction, and parity-oriented validation is maintained.

The fifth suggestion is continued enhancement of administrative monitoring and operational support. Over time, if internal adoption increases, the platform could benefit from richer operational visibility for replication health, automation status, and report-readiness monitoring, so long as these additions remain aligned with the platform's company-controlled reporting purpose.

These future suggestions should not be read as evidence that the delivered platform is incomplete in its core purpose. Rather, they reflect the natural next steps for a system that has already reached a usable and validated internal reporting state and now provides a foundation for further controlled development.

6.5 Summary

This chapter concluded the final report by evaluating the Marrybrown Sales and Payment Analytics Platform in terms of its contribution, achievement, remaining constraints, and future suggestions. The project successfully delivered a deployed internal reporting platform that reconstructed and validated selected vendor-aligned sales and payment reports, supported operational reporting access through a web portal, and provided controlled administrative functions for replication and report-readiness management.

At the same time, the project also documented real constraints, especially those related to replicated-data trustworthiness, black-box vendor logic, unresolved profit-related report behaviour, and uneven performance across some broad report scopes.

These limitations do not negate the platform's contribution; rather, they define the realistic boundary within which the delivered system should be understood.

Overall, the project achieved its core purpose as an enterprise reporting continuity and reconstruction effort conducted in a real company environment. It delivered a company-usable internal reporting platform, validated a substantial retained reporting surface, and established a practical foundation for further extension and refinement where future business needs require it.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM, 53*(4), 50-58. https://doi.org/10.1145/1721654.1721672

Azzabi, S., Alfughi, Z., & Ouda, A. (2024). Data lakes: A survey of concepts and architectures. **Computers, 13***(7), 183. https://doi.org/10.3390/computers13070183

Badger, L., Grance, T., Patt-Corner, R., & Voas, J. (2012). **Cloud computing synopsis and recommendations** (NIST Special Publication 800-146). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-146

Batini, C., & Scannapieco, M. (2006). **Data quality: Concepts, methodologies and techniques**. Springer. https://doi.org/10.1007/3-540-33173-5

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). **Manifesto for agile software development**. Agile Manifesto. https://agilemanifesto.org/

Chikofsky, E. J., & Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. **IEEE Software, 7***(1), 13–17. https://doi.org/10.1109/52.43044

FastAPI. (n.d.). **FastAPI documentation**. Retrieved March 2, 2026, from https://fastapi.tiangolo.com/

Few, S. (2012). *Use-based types of quantitative display*. Visual Business Intelligence Newsletter. Perceptual Edge. https://www.perceptualedge.com/articles/visual_business_intelligence/types_of_quantitative_display.pdf

Fielding, R. T. (2000). **Architectural styles and the design of network-based software architectures** (Doctoral dissertation, University of California, Irvine). https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

Inmon, W. H. (2005). **Building the data warehouse** (4th ed.). Wiley.

International Organization for Standardization. (2008). **ISO/IEC 25012:2008 - Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model**. https://www.iso.org/standard/35736.html

Jansen, W., & Grance, T. (2011). **Guidelines on security and privacy in public cloud computing** (NIST Special Publication 800–144). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-144

Kimball, R., & Caserta, J. (2004). **The data warehouse ETL toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data**. Wiley.

Kimball, R., & Ross, M. (2013). **The data warehouse toolkit: The definitive guide to dimensional modeling** (3rd ed.). Wiley.

Kleppmann, M. (2017). **Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems**. O'Reilly Media.

Mell, P., & Grance, T. (2011). **The NIST definition of cloud computing** (NIST Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800–145

Microsoft. (2026). *Paginated reports in the Power BI service*. Microsoft Learn. https://learn.microsoft.com/en-us/power-bi/explore-reports/end-user-paginated-report

Myers, G. J., Sandler, C., & Badgett, T. (2011). **The art of software testing** (3rd ed.). Wiley.

Oracle. (2023). *Oracle MICROS Reporting and Analytics user guide* (Release 9.0). https://docs.oracle.com/cd/E71871_01/doc.90/e72023.pdf

Pressman, R. S., & Maxim, B. R. (2014). **Software engineering: A practitioner's approach** (8th ed.). McGraw-Hill Education.

React. (n.d.). **React documentation**. Retrieved March 2, 2026, from https://react.dev/

Schwaber, K., & Sutherland, J. (2020, November). **The Scrum guide: The definitive guide to Scrum: The rules of the game**. Scrum Guides. https://scrumguides.org/scrum-guide.html

Shneiderman, B. (1996). *The eyes have it: A task by data type taxonomy for information visualizations*. In *Proceedings of the IEEE Symposium on Visual Languages* (pp. 336-343). https://www.cs.umd.edu/~ben/papers/Shneiderman1996eyes.pdf

Sommerville, I. (2015). **Software engineering** (10th ed.). Pearson.

Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. **Journal of Management Information Systems, 12**(4), 5–33. https://doi.org/10.1080/07421222.1996.11518099

Appendix

Appendix A: Report Specifications

This appendix summarises the report specifications for the Marrybrown Sales and Payment Analytics Platform. The formal implementation and business scope comprised 19 sales and payment reports. Of these, 16 reports were implemented and validated for the retained final scope, while 3 reports were not fully closed after reverse engineering and parity validation against the vendor portal. In addition, 3 customised reports were implemented based on company requests beyond the formal 19-report scope.

A.1 Formal Implementation and Business Scope Report Index

ID	Report	Report group	Final status	Notes
R01	Sale Delivery (By Sales Type) Ex Tax Calculation	Delivery and ordering channels	Implemented and validated	Delivered as a vendor-aligned delivery sales and ex-tax reporting module.
R02	Payment Type (All Payment)	Payment and voucher reconciliation	Implemented and validated	Delivered as a payment-method reconciliation report.
R03	Product Mix Report	Product, stock, and cost analysis	Not fully closed	Reverse engineering and parity validation were not sufficient to isolate the

				remaining profit logic required for final closure.
R04	Delivery - FoodPanda, Grabfood, ShopeeFood	Delivery and ordering channels	Implemented and validated	Delivered as a delivery-channel sales reporting module.
R05	Pickup & Declaration Report	Sales and exception control	Implemented and validated	Delivered as a session-level sales, declaration, and collection control report.
R06	Stock Variance Report (Latest)	Product, stock, and cost analysis	Implemented and validated	Delivered as an inventory variance monitoring report.
R07	Discount Remark Report	Discount monitoring	Not fully closed	Reverse engineering and parity validation were not sufficient to isolate the remaining profit logic required for final closure.
R08	Foodpanda Sales	Delivery and ordering channels	Implemented and validated	Delivered as a channel-specific item-

				level sales report.
R09	DELETED Items Report	Sales and exception control	Implemented and validated	Delivered as a void-item audit report.
R10	Sales Return Report	Sales and exception control	Implemented and validated	Delivered as an item-return review and reconciliation report.
R11	[SOK] Each Kiosk Transaction Report	Sales and exception control	Implemented and validated	Delivered as a kiosk transaction monitoring report.
R12	Sales Cancelled Report	Sales and exception control	Implemented and validated	Delivered as a cancelled-sales audit report.
R13	Xilnex - Monthly Checking - COGS by Item (By Sales Type)	Product, stock, and cost analysis	Implemented and validated	Delivered as an item-level monthly cost-of-goods checking report.
R14	Foodpanda Discount	Delivery and ordering channels	Implemented and validated	Delivered as a discount-focused Foodpanda reporting module.
R15	Mobile Ordering Sales	Delivery and ordering channels	Implemented and validated	Delivered as a mobile-

				ordering sales report.
R16	Average SOS Report (New)	Service and operational timing	Implemented and validated	Delivered as a store-level service-duration monitoring report.
R17	MB Cash Voucher (with Barcode) Redemption Report	Payment and voucher reconciliation	Implemented and validated	Delivered as a barcode voucher redemption report.
R18	MB Staff E - Voucher RM 20 & MB CASH VOUCHER RM10 (with Barcode) Redemption Report	Payment and voucher reconciliation	Implemented and validated	Delivered as a staff and cash voucher redemption control report.
R19	Product Mix with modifier without ETL	Product, stock, and cost analysis	Not fully closed	Reverse engineering and parity validation were not sufficient to isolate the remaining profit logic required for final closure.

Table A.1: Formal implementation and business scope report index

A.2 Additional Customised Reports

ID	Customised report	Reporting purpose	Final status
C01	Roblox Free Chicken Burger Combo Sales	Tracks a company-requested promotional sales scenario for campaign monitoring and checking.	Implemented and validated
C02	Voucher Campaign & Reward Sales	Tracks campaign- and reward-related voucher sales activity requested during implementation.	Implemented and validated
C03	Promotion Item Additional Purchase Report	Analyses additional-purchase behaviour for selected promotion items requested by the company.	Implemented and validated

Table A.2: Additional customised reports implemented beyond the formal 19-report scope

A.3 Detailed Report Specifications for the Formal 19-Report Scope

R01: Sale Delivery (By Sales Type) Ex Tax Calculation

Field	Description
Report name	Sale Delivery (By Sales Type) Ex Tax Calculation
Business purpose	To analyse delivery-related sales by sales type and delivery channel while presenting tax-exclusive values required for finance checking and channel reconciliation.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status and payment status filters; optional advanced search conditions where detailed transaction filtering is needed.
Output summary	Returns grouped rows by date, store, sales type, sales return type, and delivery type, with transaction quantity, sales totals, tax-exclusive amounts, tax amount, payment-bucket amounts, and summary totals.
Data scope	Replicated sales records, payment records, location data, and supporting customer-search metadata where relevant to advanced search.
Key business rules	Delivery and payment metrics are reconstructed through merged sales-side and payment-side logic; grouping follows vendor-aligned delivery hierarchy; monetary totals are normalised after aggregation; all-zero rows are suppressed from the final output.
Validation approach	Report output was validated through parity comparison against vendor portal exports using matched reporting windows and filter selections.
Final status	Implemented and validated.

Table A.3: Detailed specification for R01

R02: Payment Type (All Payment)

Field	Description
Report name	Payment Type (All Payment)
Business purpose	To provide a detailed breakdown of sales collections by payment method for reconciliation, outlet review, and month-end checking.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status filter; optional payment status filter.
Output summary	Returns rows grouped by store, order source, payment method, device, card type, and reference, with total collection and payment-bucket amounts.
Data scope	Replicated payment records, sales records, and outlet reference data.
Key business rules	Payment rows are grouped to reproduce portal payment presentation; payment-method categories are normalised into cash, card, voucher, e-wallet, and other buckets; return sales and non-qualifying records are excluded according to report rules.
Validation approach	Report output was validated through parity comparison against vendor portal exports across selected date ranges and store scopes.
Final status	Implemented and validated.

Table A.4: Detailed specification for R02

R03: Product Mix Report

Field	Description
Report name	Product Mix Report

Business purpose	To present item-level product sales composition for management review, quantity checking, and profit-related analysis.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); expected outlet scope and item-related optional filters consistent with product-level reporting requirements.
Output summary	Intended to return item-level grouped sales rows with quantity, value, and profit-related measures.
Data scope	Replicated sales records, sales item records, item master data, and supporting reference data required for product-level grouping and profit-related calculations.
Key business rules	The report depends on correct product-level grouping, item classification, and profit-related derivation consistent with portal behaviour.
Validation approach	Reverse engineering and parity validation were performed against vendor portal behaviour, but the remaining profit-related logic could not be isolated conclusively within the project boundary.
Final status	Not fully closed.

Table A.5: Detailed specification for R03

R04: Delivery - FoodPanda, Grabfood, ShopeeFood

Field	**Description**
Report name	Delivery - FoodPanda, Grabfood, ShopeeFood
Business purpose	To provide delivery-channel sales visibility across the main third-party ordering platforms used in the reporting scope.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status filter; optional item-related filters such as

	type, category, group name, item division, and brand where applicable.
Output summary	Returns delivery-channel sales rows and totals aligned to platform-specific sales activity across FoodPanda, GrabFood, and ShopeeFood.
Data scope	Replicated sales records, sales item records, item master data, outlet reference data, and delivery-related sales attributes.
Key business rules	Channel classification must follow delivery-type logic consistent with portal output; item filters must behave consistently across channels; rows outside the required delivery scope are excluded.
Validation approach	Report output was validated through parity comparison against vendor portal exports for selected channels, date ranges, and outlet scopes.
Final status	Implemented and validated.

Table A.6: Detailed specification for R04

R05: Pickup & Declaration Report

Field	**Description**
Report name	Pickup & Declaration Report
Business purpose	To monitor cashier session-level sales, collection, declaration, and short/excess behaviour for operational and reconciliation control.
Primary users	Finance, Operations, and supervisory users involved in cash-control checking.
Parameters	Start date and end date (required); optional outlet scope; optional sales status filter; optional payment status filter.
Output summary	Returns session-level rows containing sales totals, payment collections, declaration amounts, short/excess values, safe-deposit totals, and register-float totals.

Data scope	Replicated sales records, payment records, register-log records, denomination-related records, and outlet/session reference data.
Key business rules	Multiple source branches are regrouped into the portal-visible session shape; short/excess is derived from internal movement and declaration totals; zero-only grouped rows are suppressed from the final output.
Validation approach	Report output was validated through strict parity comparison against vendor workbook exports for matched date ranges and status scopes.
Final status	Implemented and validated.

Table A.7: Detailed specification for R05

R06: Stock Variance Report (Latest)

Field	**Description**
Report name	Stock Variance Report (Latest)
Business purpose	To present outlet-level inventory variance by item so that stock discrepancies can be reviewed and followed up operationally.
Primary users	Operations, inventory, and Finance users.
Parameters	Start date and end date (required); optional outlet scope; optional type, category, and group-name filters; optional advanced search over item and store attributes.
Output summary	Returns grouped rows by store and item, including opening stock, received quantity, purchase return, wastage, transfer, staff meal, expected sold quantity, expected and physical closing stock, variance quantity, and variance amount.
Data scope	Replicated stock, stock-take, stock-received, sales, sales item, item master, vendor, and outlet-related records.
Key business rules	Inventory movement, stockcheck, and sold-quantity logic are combined to produce the latest variance view; grouping

	follows vendor-visible item hierarchy; advanced search is applied before final grouping.
Validation approach	Report output was validated against vendor workbook exports and cross-checked across source, replica, API, and frontend output for the validated scope.
Final status	Implemented and validated.

Table A.8: Detailed specification for R06

R07: Discount Remark Report

Field	**Description**
Report name	Discount Remark Report
Business purpose	To provide visibility into discount usage and associated remarks for checking, review, and exception follow-up.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); expected outlet scope and discount-related optional filters consistent with discount-level reporting.
Output summary	Intended to return discount-related rows with sales context, discount information, remarks, and profit-related measures.
Data scope	Replicated sales records, sales item records, discount-related fields, and supporting item and outlet reference data.
Key business rules	The report depends on accurate discount-level grouping, remark handling, and profit-related derivation consistent with portal behaviour.
Validation approach	Reverse engineering and parity validation were performed against vendor portal behaviour, but the remaining profit-related logic could not be isolated conclusively within the project boundary.
Final status	Not fully closed.

Table A.9: Detailed specification for R07

R08: Foodpanda Sales

Field	Description
Report name	Foodpanda Sales
Business purpose	To provide item-level Foodpanda sales visibility for channel-specific operational checking and reconciliation.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status filter; optional sales type, type, category, group name, item division, and brand filters.
Output summary	Returns item-level Foodpanda sales rows including date, location, customer and cashier metadata, delivery type, sales reference, quantity, and net and gross sold amounts.
Data scope	Replicated sales records, sales item records, item master data, location data, and supporting recipe or package-related reference paths needed by the validated logic.
Key business rules	Only rows within the intended Foodpanda delivery scope are retained; delivery-type and item-filter behaviour is aligned to portal semantics; row structure remains item-level for checking and export purposes.
Validation approach	Report output was validated through strict parity comparison against vendor portal exports across multiple windows and filtered scenarios.
Final status	Implemented and validated.

Table A.10: Detailed specification for R08

R09: DELETED Items Report

Field	Description
Report name	DELETED Items Report
Business purpose	To provide an audit view of deleted or voided items for operational control and exception review.

Primary users	Finance, Operations, and supervisory users.
Parameters	Start date and end date (required); optional outlet scope.
Output summary	Returns item-level void rows including location, sales number, void time, item code, item name, void person, void reason, perform type, void quantity, and void amount.
Data scope	Replicated void-item records, sales item records, sales records, item master data, and outlet reference data.
Key business rules	The report is constrained to the portal-aligned void-item scope; void quantity and amount are aggregated at the report-row level; ordering is stabilised to keep consistent row presentation.
Validation approach	Report output was validated through strict parity comparison against vendor portal export output for matched date windows.
Final status	Implemented and validated.

Table A.11: Detailed specification for R09

R10: Sales Return Report

Field	**Description**
Report name	Sales Return Report
Business purpose	To list returned items and related values for reconciliation, outlet review, and exception control.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status filter; optional item-related filters such as type, category, group name, and item division where applicable.
Output summary	Returns item-level return rows containing sales and return references, item details, quantities, and monetary return values.

Data scope	Replicated sales records, return-related sales item records, item master data, and outlet reference data.
Key business rules	Only return-scope transactions are included; item filtering and row grouping must reproduce portal behaviour; return values are presented in export-ready tabular form.
Validation approach	Report output was validated through parity comparison against vendor portal exports for matched scopes and item-filter conditions.
Final status	Implemented and validated.

Table A.12: Detailed specification for R10

R11: [SOK] Each Kiosk Transaction Report

Field	**Description**
Report name	[SOK] Each Kiosk Transaction Report
Business purpose	To present kiosk transaction records for channel monitoring, transaction review, and operational follow-up.
Primary users	Operations and Finance users.
Parameters	Start date and end date (required); optional outlet scope; optional transaction or item-related filters where applicable to kiosk-report usage.
Output summary	Returns kiosk transaction rows aligned to the required portal transaction view, including kiosk-related transaction context and monetary values.
Data scope	Replicated sales records, payment records where required, outlet reference data, and kiosk-related transactional attributes.
Key business rules	Kiosk transactions are isolated according to the defined channel logic; output retains the required transaction-level visibility for checking and export.

Validation approach	Report output was validated through parity comparison against vendor portal exports for the retained kiosk-report scope.
Final status	Implemented and validated.

Table A.13: Detailed specification for R11

R12: Sales Cancelled Report

Field	**Description**
Report name	Sales Cancelled Report
Business purpose	To provide an audit trail of cancelled sales for exception control, follow-up, and reconciliation review.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional item-related filters where applicable.
Output summary	Returns cancelled-sales rows containing outlet, date, sales reference, quantity, sold value, and cancellation-related details such as cancelling user and remarks.
Data scope	Replicated sales records, cancellation metadata, item-related records where required by filtering, and outlet reference data.
Key business rules	Only cancelled or voided sales within the selected date scope are included; output ordering and textual normalisation follow portal display behaviour.
Validation approach	Report output was validated through parity comparison against vendor portal exports for matched scopes.
Final status	Implemented and validated.

Table A.14: Detailed specification for R12

R13: Xilnex - Monthly Checking - COGS by Item (By Sales Type)

Field	Description
Report name	Xilnex - Monthly Checking - COGS by Item (By Sales Type)
Business purpose	To support monthly checking of item-level cost of goods sold by sales type for Finance review and cost monitoring.
Primary users	Finance users.
Parameters	Start date and end date (required); optional outlet scope; optional sales type and item-related filters where applicable.
Output summary	Returns item-level rows grouped by sales type, with quantity, value, and cost-related fields required for monthly COGS checking.
Data scope	Replicated sales records, sales item records, item master data, cost-related attributes, and outlet reference data.
Key business rules	Cost and sales-type grouping must remain consistent with the validated monthly-checking view; item-level output is retained for checking rather than reduced to a dashboard summary.
Validation approach	Report output was validated through parity comparison against vendor portal exports for the retained monthly-checking scope.
Final status	Implemented and validated.

Table A.15: Detailed specification for R13

R14: Foodpanda Discount

Field	Description
Report name	Foodpanda Discount
Business purpose	To provide discount-level visibility for Foodpanda-related transactions so that channel discount behaviour can be reviewed and reconciled.

Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status and item-related filters where applicable.
Output summary	Returns Foodpanda discount rows and totals aligned to the portal's discount-focused reporting view for the channel.
Data scope	Replicated sales records, sales item records, discount-related fields, item master data, and outlet reference data.
Key business rules	Only Foodpanda-scope transactions are included; discount grouping and item-filter behaviour must remain aligned to the validated channel-report rules.
Validation approach	Report output was validated through parity comparison against vendor portal exports for matched Foodpanda reporting windows.
Final status	Implemented and validated.

Table A.16: Detailed specification for R14

R15: Mobile Ordering Sales

Field	**Description**
Report name	Mobile Ordering Sales
Business purpose	To provide sales visibility for mobile-ordering transactions as a distinct ordering channel within the reporting platform.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional sales status and item-related filters where applicable.
Output summary	Returns mobile-ordering sales rows and totals aligned to the required portal reporting view for the channel.
Data scope	Replicated sales records, sales item records, item master data, outlet reference data, and ordering-channel attributes.
Key business rules	Channel classification must distinguish mobile-ordering sales from other sales types; report output remains export-oriented and transaction-traceable.

Validation approach	Report output was validated through parity comparison against vendor portal exports for matched mobile-ordering scopes.
Final status	Implemented and validated.

Table A.17: Detailed specification for R15

R16: Average SOS Report (New)

Field	**Description**
Report name	Average SOS Report (New)
Business purpose	To monitor average service-order-speed behaviour at store level for operational review of order preparation performance.
Primary users	Operations users, with secondary management and Finance use where required for service-performance review.
Parameters	Start date and end date (required); optional outlet scope; optional sales status and sales type filters; optional single-rule advanced search on selected sales-item identifiers.
Output summary	Returns store-level rows containing order count and preparation-duration measures for the selected reporting scope.
Data scope	Replicated KDS-related records, sales records, sales item records, item reference data where required by advanced search, and outlet reference data.
Key business rules	Preparation duration is derived from KDS order and collection timestamps; item-level timing rows are first grouped to order level and then rolled up to store level using the validated weighting logic; unsupported portal-option filters remain excluded from active analytical use.

Validation approach	Report output was validated through parity comparison against vendor portal output for the retained report scope.
Final status	Implemented and validated.

Table A.18: Detailed specification for R16

R17: MB Cash Voucher (with Barcode) Redemption Report

Field	**Description**
Report name	MB Cash Voucher (with Barcode) Redemption Report
Business purpose	To track redemption of barcode-based cash vouchers for voucher control and reconciliation.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); optional outlet scope; optional voucher-variant inclusion logic where required by the report behaviour.
Output summary	Returns voucher redemption rows including redemption date, outlet, voucher identifier, voucher category, linked sales reference, and reconciliation-related totals or counts.
Data scope	Replicated voucher-related payment or redemption records, linked sales records, and outlet reference data.
Key business rules	Voucher identifiers and categories are normalised to match portal grouping; duplicated joins are suppressed so that redemption rows remain parity-aligned.
Validation approach	Report output was validated through parity comparison against vendor portal exports using matched date ranges and outlet scopes.
Final status	Implemented and validated.

Table A.19: Detailed specification for R17

R18: MB Staff E - Voucher RM 20 & MB CASH VOUCHER RM10 (with Barcode) Redemption Report

Field	Description
Report name	MB Staff E - Voucher RM 20 & MB CASH VOUCHER RM10 (with Barcode) Redemption Report
Business purpose	To monitor redemption of staff e-vouchers and selected barcode cash vouchers for controlled checking and reconciliation.
Primary users	Finance and Operations users, with secondary administrative interest where staff-benefit monitoring is required.
Parameters	Start date and end date (required); optional outlet scope; optional voucher-variant inclusion logic where required by the report behaviour.
Output summary	Returns voucher redemption rows including redemption date, outlet, voucher identifier, voucher type, linked sales reference, and reconciliation-related totals or counts.
Data scope	Replicated voucher-related payment or redemption records, linked sales records, and outlet reference data.
Key business rules	Staff and cash voucher variants must be grouped consistently with portal behaviour; duplicated rows are suppressed and legacy variations are normalised into the retained voucher categories.
Validation approach	Report output was validated through parity comparison against vendor portal exports for matched scopes.
Final status	Implemented and validated.

Table A.20: Detailed specification for R18

R19: Product Mix with modifier without ETL

Field	Description
Report name	Product Mix with modifier without ETL
Business purpose	To extend product-mix analysis with modifier-level detail for checking sales composition and profit-related behaviour at a more detailed reporting level.
Primary users	Finance and Operations users.
Parameters	Start date and end date (required); expected outlet scope and item-related optional filters consistent with modifier-level product reporting.
Output summary	Intended to return modifier-aware product-mix rows with quantity, value, and profit-related measures.
Data scope	Replicated sales records, sales item records, modifier-related transactional fields, item master data, and supporting reference data needed for modifier-level grouping.
Key business rules	The report depends on correct modifier-level grouping and profit-related derivation consistent with portal behaviour.
Validation approach	Reverse engineering and parity validation were performed against vendor portal behaviour, but the remaining profit-related logic could not be isolated conclusively within the project boundary.
Final status	Not fully closed.

Table A.21: Detailed specification for R19

A.4 Detailed Specifications for Additional Customised Reports

C01: Roblox Free Chicken Burger Combo Sales

Field	Description
Report name	Roblox Free Chicken Burger Combo Sales

Business purpose	To track a campaign-specific promotional sales scenario requested by the company for monitoring and follow-up.
Primary users	Finance, Operations, and campaign-monitoring users.
Parameters	Start date and end date (required); optional outlet scope; optional campaign-related or transaction filters where applicable to the implemented report view.
Output summary	Returns campaign-related sales rows and totals aligned to the required promotional sales scenario.
Data scope	Replicated sales records, sales item records, and supporting item and outlet reference data required for the campaign-specific grouping.
Key business rules	Campaign scope is isolated through report-specific selection logic while reusing the shared platform pattern of parameter handling, tabular output, and export-oriented delivery.
Validation approach	Report output was validated through comparison against the required business expectations and retained portal-aligned checking workflow for the customised scope.
Final status	Implemented and validated.

Table A.22: Detailed specification for C01

C02: Voucher Campaign & Reward Sales

Field	**Description**
Report name	Voucher Campaign & Reward Sales
Business purpose	To provide campaign- and reward-related voucher sales visibility requested by the company beyond the formal report scope.
Primary users	Finance, Operations, and campaign-monitoring users.
Parameters	Start date and end date (required); optional outlet scope; optional campaign or voucher-related filters where applicable.
Output summary	Returns voucher campaign and reward sales rows and totals aligned to the required customised reporting view.

Data scope	Replicated sales records, payment or voucher-related records, and supporting outlet and reference data required for campaign grouping.
Key business rules	Campaign and reward classification logic is applied within the shared semantic layer while preserving the same parameter, export, and tabular reporting pattern used across the main platform.
Validation approach	Report output was validated through comparison against required business expectations and retained checking artefacts for the customised reporting scope.
Final status	Implemented and validated.

Table A.23: Detailed specification for C02

C03: Promotion Item Additional Purchase Report

Field	**Description**
Report name	Promotion Item Additional Purchase Report
Business purpose	To analyse additional-purchase behaviour associated with selected promotion items requested by the company.
Primary users	Finance, Operations, and business users monitoring promotional sales behaviour.
Parameters	Start date and end date (required); optional outlet scope; optional promotion-item or transaction filters where applicable to the implemented view.
Output summary	Returns rows and totals showing additional-purchase behaviour linked to the defined promotion-item scope.
Data scope	Replicated sales records, sales item records, and supporting item and outlet reference data needed for promotion-item grouping and comparison.
Key business rules	The report uses report-specific promotion-item selection logic while retaining the same shared platform pattern of parameter-driven retrieval, tabular results, and export-ready output.

Validation approach	Report output was validated through comparison against required business expectations and retained checking artefacts for the customised scope.
Final status	Implemented and validated.

Table A.24: Detailed specification for C03