

INDIVIDUAL THESIS

SMART ORDER RECOMMENDATION SYSTEM FOR THE SMARTCOST POS PLATFORM

Jayani Dasanayaka

(IT22152664)

Individual component: Smart Order Recommendation System

Group ID: R26-IT-026

Report submitted in partial fulfilment of the requirements for the

BSc (Hons) Degree in Information Technology

Specialising in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

October 2026

DECLARATION AND SUPERVISOR CERTIFICATION

Candidate declaration. I declare that this thesis is my own work and does not incorporate, without acknowledgement, material previously submitted for a degree or diploma at any university or institute of higher learning. To the best of my knowledge, it contains no material previously published or written by another person except where acknowledgement is made in the text. I grant the Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and distribute this thesis, in whole or in part, in print, electronic, or other media, while retaining the right to use the content in future work.

Academic integrity and AI disclosure. I have disclosed all use of generative AI and other automated tools in accordance with the module policy. I remain responsible for the accuracy, originality, citations, code, data, analyses, and conclusions in this thesis.

Student name: Jayani Dasanayaka

Student ID: IT2215266

Signature: _____

Date: __09/10/2026_____

Supervisor certification. The above candidate carried out this research for the bachelor's degree thesis under my supervision.

Role and name	Signature	Date
Supervisor: Mr. Deemantha Siriwardhana		
Co-supervisor: Mrs. Ayesha Wijesooriya		

Abstract

Small restaurants and shops record every sale in a point-of-sale (POS) system, but the history is usually used only for billing and reports. This thesis describes the Smart Order Recommendation System, my individual component of the SmartCost POS platform. It turns past orders into product suggestions that appear while a customer is still building an order. Each completed order at a branch is treated as a market basket. Apriori association rules are mined from these baskets and combined with item-item cosine similarity, category attach rates, overall popularity and, when a customer supplies a mobile number, personal favourites. Products that are out of stock or already in the cart are removed before ranking, and every suggestion carries a short reason in plain language. A Python/FastAPI service does the analysis. A Node.js/Express backend with MongoDB prepares the data, a Next.js Quick Response (QR) ordering page shows suggestions to customers, and an Expo/React Native screen shows buy-together rules to staff. I checked the pipeline with a deterministic synthetic dataset of 12,000 orders containing planted co-purchase patterns, so the mining and ranking could be compared with known answers, and I exercised the application programming interface (API) and both interfaces end to end in the development environment. The component returns ranked, explainable recommendations and exposes model statistics, rules and category attach rates to staff. The QR ordering page has not yet been deployed publicly. The main limitations are that the model is re-mined on every request, thresholds and weights are fixed, new products have little evidence, accepted suggestions are not stored, and no hold-out Precision@K or Hit Rate has been measured. These limitations are set out as future work.

Keywords: Smart Order Recommendation, Point-of-Sale, Association Rule Mining, Hybrid Recommender, Micro-SMEs

ACKNOWLEDGEMENTS

I would like to thank my supervisors for their guidance, feedback and patience throughout this research. Their comments helped me narrow the research problem, tighten the technical design and describe my own contribution honestly.

I also thank the other members of the SmartCost research group. Without their work on the shared backend, the database, the customer ordering workflow and the staff application, the recommendation component would have had nothing to plug into. I am grateful to the academic staff of the Sri Lanka Institute of Information Technology for the teaching and resources I received during the degree programme.

Finally, I thank my family, friends and colleagues for their encouragement while I completed this thesis.

TABLE OF CONTENTS

DECLARATION AND SUPERVISOR CERTIFICATION.....	i
Abstract.....	ii
ACKNOWLEDGEMENTS.....	iii
TABLE OF CONTENTS.....	iv
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ABBREVIATIONS	x
LIST OF APPENDICES	xi
1 INTRODUCTION.....	1
1.1 Overall Project Context	1
1.2 Individual Component Context	3
1.3 Problem Statement	5
1.4 Research Gap.....	5
1.5 Aim and Objectives	6
1.6 Research Questions or Hypotheses	7
1.7 Scope, Limitations and Assumptions	8
1.8 Individual Contribution	9
1.9 Thesis Structure.....	11
2 LITERATURE REVIEW AND RELATED WORK	13
2.1 Review Method and Scope.....	13
2.2 Domain and Theoretical Foundations	14
2.2.1 POS transaction data as behavioural evidence	14
2.2.2 Market basket analysis and frequent itemsets.....	15
2.2.3 Association rules: support, confidence and lift.....	15
2.2.4 Item-item collaborative filtering and cosine similarity.....	16
2.2.5 Hybrid recommendation and score fusion	16
2.2.6 Explainability, diversity and availability constraints.....	17
2.3 Related Research	18
2.4 Existing Systems, Products and Techniques	19
2.5 Gap Synthesis and Conceptual Framework.....	21
3 RESEARCH METHODOLOGY	25
3.1 Research Design	25
3.2 Data, Participants or Study Materials.....	27
3.3 Experimental or Study Procedure.....	29

3.4 Measures and Evaluation Metrics	31
3.5 Data Analysis	33
3.6 Ethics, Privacy, Security and Responsible AI.....	35
3.7 Reproducibility and Research Artefacts.....	36
4 PRODUCT AND SOLUTION DESIGN.....	39
4.1 Requirements.....	39
4.1.1 Functional requirements	39
4.1.2 Non-functional and quality requirements	40
4.2 Component Architecture within the Complete Solution	41
4.3 Design Alternatives and Decisions.....	43
4.4 Detailed Design	44
4.4.1 Request and context design	45
4.4.2 Data preparation design	45
4.4.3 Recommendation signal design	46
4.4.4 Output and interface design.....	47
4.4.5 API design	48
4.5 Standards and Professional Practice.....	48
5 IMPLEMENTATION, INTEGRATION AND TESTING	50
5.1 Development Environment and Technology Stack.....	50
5.2 Component Implementation	52
5.2.1 Data preparation and candidate construction.....	53
5.2.2 Hybrid recommendation engine.....	53
5.2.3 Output enrichment and explainability.....	55
5.3 Integration with the Other Components	55
5.3.1 Customer QR ordering integration	56
5.3.2 Staff mobile application integration	57
5.3.3 Integration with manual orders and billing.....	58
5.4 Verification and Software Testing	61
5.4.1 Algorithm and ranking verification	62
5.4.2 API verification	63
5.4.3 Interface and end-to-end integration testing	64
5.5 Deployment and Operation	64
6 RESULTS AND VALIDATION.....	67
6.1 Research Results.....	67
6.2 Product and Technical Results	70
6.3 User or Stakeholder Validation	71
6.4 Integration Results.....	73
6.5 Summary of Findings	75
7 DISCUSSION	77

7.1 Interpretation of Findings	78
7.2 Comparison with Literature Baselines and Existing Solutions	80
7.3 Research Contribution	81
7.4 Product Contribution	83
7.5 Limitations and Threats to Validity	84
7.6 Implications	86
8 COMMERCIALISATION AND SUSTAINABILITY	89
8.1 Value Proposition and Target Users	89
8.2 Market and Competitive Position	90
8.3 Business or Sustainability Model	91
8.4 Scalability, Intellectual Property and Compliance	92
8.5 Contribution to the Sustainable Development Goals	94
9 CONCLUSION AND FUTURE DIRECTIONS	95
9.1 Conclusion	95
9.2 Achievement of Objectives	96
9.3 Individual Research and Product Contributions	97
9.4 Future Directions	98
REFERENCES	100
GLOSSARY	102
APPENDICES	104
Appendix A: Research Instruments or Extended Results	104
A.1 Deterministic seed parameters and planted relationships	104
A.2 Core request parameters	105
A.3 Documented model outputs	105
A.4 Evaluation dimensions reported in the research paper	105
Appendix B: Academic Integrity and AI Use Disclosure	106

LIST OF FIGURES

Figure No.	Figure Title	Page
Figure 1.1	Position of the recommendation component within the SmartCost POS solution	3
Figure 1.2	Implemented recommendation interfaces: QR menu, customer suggestion dialog and staff recommendation insights	4
Figure 2.1	Conceptual framework for the Smart Order Recommendation System	23
Figure 3.1	Research methodology pipeline for the Smart Order Recommendation System	26
Figure 3.2	Validation and reproducibility workflow	38
Figure 4.1	Smart Order Recommendation System within the complete SmartCost POS solution	42
Figure 4.2	Detailed recommendation processing flow	45
Figure 5.1	Implemented request-to-recommendation execution path	55
Figure 5.2	Customer recommendation dialog in the QR ordering workflow	56
Figure 5.3	Customer order review stage after recommendation interaction	57
Figure 5.4	Staff recommendation model summary screen	58
Figure 5.5	Staff view of learned recommendation patterns	58
Figure 5.6	SmartCost mobile interface entry point for manual orders	59
Figure 5.7	QR code screen used within the manual-order workflow	60
Figure 5.8	Manual-order list before accepting an order	61

Figure 5.9	Confirmation after accepting and opening the order	61
Figure 6.1	Ground-truth validation strategy for the Smart Order Recommendation System	69
Figure 6.2	Integrated results across SmartCost application layers	74
Figure 7.1	Interpretive synthesis of findings, contributions and current boundaries	78
Figure 7.2	Main threats to validity and corresponding mitigation directions	85
Figure 6.1	Ground-truth validation strategy for the Smart Order Recommendation System	69
Figure 6.2	Integrated results across SmartCost application layers	74
Figure 7.1	Interpretive synthesis of findings, contributions and current boundaries	78

LIST OF TABLES

Table No.	Table Title	Page
Table 1.1	Main system layers related to the Smart Order Recommendation System	3
Table 1.2	Research gap and the corresponding response of the proposed component	6
Table 1.3	Research and product objectives of the Smart Order Recommendation System	7
Table 1.4	Research questions guiding the individual component	7
Table 1.5	Scope boundaries and assumptions	9
Table 1.6	Summary of the individual contribution	10
Table 1.7	Organisation of the remaining thesis	11
Table 2.1	Scope of the literature review	14
Table 2.2	Interpretation of association-rule measures	16
Table 2.3	Comparison of recommendation techniques relevant to small-business POS environments	20
Table 2.4	Literature-derived design requirements mapped to the Smart Order Recommendation System	21
Table 2.5	Synthesis of research gaps and design responses	22
Table 3.1	Research design summary	27
Table 3.2	Data and study materials	29
Table 3.3	Core recommendation parameters used in the study procedure	29
Table 3.4	Measures and evaluation metrics	33
Table 3.5	Data-analysis stages	34
Table 3.6	Ethics, privacy, security and responsible-AI considerations	36
Table 3.7	Reproducible research artefacts	38
Table 4.1	Functional requirements of the Smart Order Recommendation System	40
Table 4.2	Non-functional and quality requirements	41

Table 4.3	Major components and responsibilities	42
Table 4.4	Design alternatives and implemented decisions	43
Table 4.5	Recommendation signals used in score fusion	46
Table 4.6	Explainable recommendation reason mapping	47
Table 4.7	API boundaries of the recommendation component	48
Table 5.1	Development environment and technology stack	51
Table 5.2	Main implementation artefacts	52
Table 5.3	Deterministic test-data configuration	62
Table 5.4	Planted patterns used for behavioural validation	62
Table 5.5	Representative API verification cases	63
Table 5.6	Current operational state and deployment improvements	65
Table 6.1	Research outputs supported by the current implementation	68
Table 6.2	Observable technical outputs of the recommendation component	71
Table 6.3	Stakeholder validation evidence and interpretation	72
Table 6.4	Integration validation summary	74
Table 7.1	Interpretation of the principal findings	79
Table 7.2	Comparison with common recommendation approaches	81

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
FR	Functional Requirement
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
ISO	International Organization for Standardization
JWT	JSON Web Token
KPI	Key Performance Indicator
NFR	Non-Functional Requirement
PCI DSS	Payment Card Industry Data Security Standard
POS	Point-of-Sale
QR	Quick Response
REST	Representational State Transfer
RQ	Research Question
SDG	Sustainable Development Goal
SLIIT	Sri Lanka Institute of Information Technology
SME	Small and Medium-sized Enterprise
UI	User Interface

LIST OF APPENDICES

Appendix	Description	Page
Appendix A	Seed parameters, request parameters, model outputs and evaluation dimensions	104
Appendix B	AI tools used, purpose, verification and affected sections	106

1 INTRODUCTION

Most small shops and restaurants now record their sales in a point-of-sale (POS) system. The system prints the bill, updates stock and keeps a history of what was sold. That history is worth more than many owners realise. When completed orders are read together they show which dishes tend to be bought together, which kind of item is often missing from a meal, and what a returning customer keeps ordering [1], [2]. In this thesis I describe how I used that history to build product suggestions inside SmartCost, a POS platform aimed at micro and small businesses, the kind often called micro-SMEs (small and medium-sized enterprises, SMEs).

The Smart Order Recommendation System is my individual component of SmartCost. Its job is narrow. While a customer is still building an order, it works out which extra items are likely to suit that order and shows them before the order is sent. I did not want it to be an isolated prototype, so it runs inside the real ordering flow. A customer scans a Quick Response (QR) code, opens the branch menu and adds items to a cart. At the review step a dialog titled “Add something else?” appears with a short list of suggestions. The same learned patterns are also shown to restaurant staff in the mobile application, where they can read the buy-together rules and category attach rates behind the suggestions.

The recommender is a hybrid. Apriori association rules find items that are bought together often enough to matter [3]. Item-item cosine similarity picks up weaker links that fall below the rule threshold [4], [5]. Category attach rates show which kind of item is missing from the cart, favourites bring in a customer’s own history when a mobile number is supplied, and overall popularity acts as a weak fallback. Every signal is scaled to the same range, weighted and added into a single score, and a small diversity rule stops the final list from repeating the same kind of reason.

1.1 Overall Project Context

SmartCost is meant to be a practical POS and analytics platform for micro and small enterprises. The wider solution covers everyday operations such as menu and product management, carts and billing, order handling and stock-aware product availability. It

also covers analytics that turn accumulated business data into decision support. The recommendation component sits at one specific moment where analytics can change an operational decision: the point where a customer has chosen some products but has not yet finalised the order.

Recommending products in a POS setting is not the same as recommending them in a large online marketplace. A restaurant or small outlet has a short catalogue, modest infrastructure and few interactions per customer, and it needs a fast answer. The recommender therefore has to use data the business already holds, work without customer accounts, respect what is actually in stock at the branch, and respond quickly enough not to slow ordering down. The current design handles this by using completed orders from the previous 365 days, grouping them by shop and branch, and using a customer's mobile number only as an optional extra signal.

The SmartCost architecture keeps the transactional system and the analytical service apart. Figure 1.1 shows where the recommender sits. The ordering page is written in Next.js, the operational backend in Node.js/Express, the data layer uses MongoDB, and the analytics run in a Python/FastAPI service. Node.js prepares the branch baskets, the orderable catalogue, the customer favourites and the cart categories. The Python service mines patterns, scores candidates and ranks them. Node.js then adds the name, price, image, category and reason text to each returned product identifier. Because of this split, the analytical logic can change without touching the code that handles transactions. Table 1.1 lists the layers and what each does for the recommender.

SmartCost Recommendation Component in the Complete POS Solution

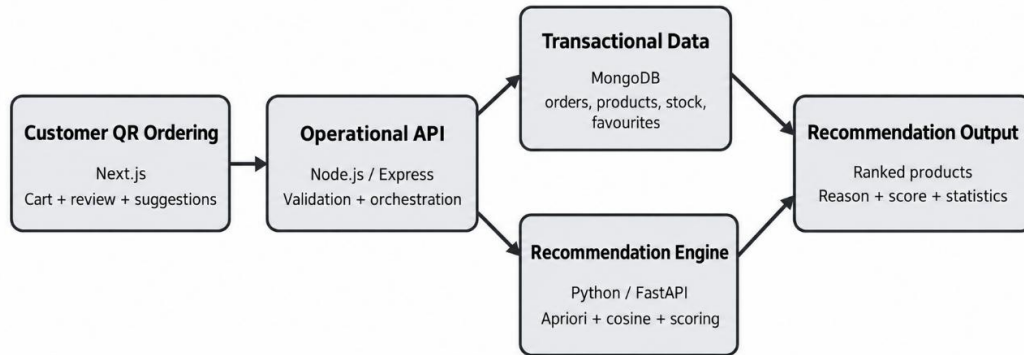


Figure 1.1: Position of the recommendation component within the SmartCost POS solution

Table 1.1: Main system layers related to the Smart Order Recommendation System

Layer	Technology / artefact	Role in the recommendation workflow
Customer ordering	Next.js QR ordering page	Builds the cart, requests recommendations before final submission, and lets the customer add or skip suggestions.
Operational backend	Node.js / Express	Validates branch and cart data, prepares baskets and candidates, calls the analysis service and formats customer-facing explanations.
Data layer	MongoDB / Mongoose	Stores historical orders, products, branch stock and customer purchase records used to prepare the inputs.
Analysis service	Python / FastAPI	Runs Apriori mining, association rules, item-item cosine similarity, category attach analysis, score fusion and ranking.
Staff application	Expo / React Native	Shows model statistics, buy-together rules, category attach rates and other insights to staff.

1.2 Individual Component Context

The Smart Order Recommendation System is the recommendation component of SmartCost. It looks for products that suit the customer’s current basket and presents them before the order is finalised. A plain “popular items” widget would show the same products to everyone. This component tries to say why an item fits this particular basket, with reasons such as “Often ordered with Chicken Kottu”, “Goes well with Chicken Kottu”, “Most loved Desserts”, “Popular right now” and “You ordered this before”. The wording comes from whichever signal contributes most to the final score.

The component has two audiences, and Figure 1.2 shows the screens for both. The first is the customer on the QR ordering page. After choosing products and tapping the button to review and send the order, the customer sees suggestions with product details and controls to add or remove them. The second audience is the restaurant staff or owner. The staff Recommendations screen runs the same model in an insight mode and shows summary statistics, learned buy-together patterns and category attach rates. One learned structure therefore serves two purposes: it supports upselling at the moment of ordering, and it helps staff understand what their customers tend to buy together.

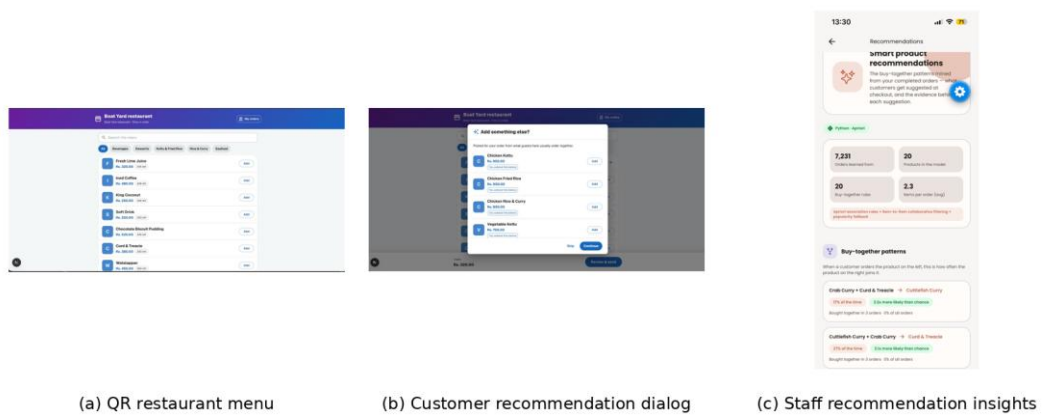


Figure 1.2: Implemented recommendation interfaces: QR menu, customer suggestion dialog and staff recommendation insights

In the working walkthrough, a customer picks items from the menu and sees the suggestion dialog before continuing. Later, staff can search for the order and handle it through the manual-order screens and the QR flow. Once a manual order is accepted and opened, its items appear in the cart and the normal billing steps carry on, including the existing kitchen and counter processes. The recommender plugs into the established order lifecycle and does not replace the POS checkout.

One point needs to be stated clearly. The QR ordering workflow is fully implemented and has been tested in the development environment, but it has not yet been deployed to a public hosting service. Throughout this thesis I separate “functionally implemented” from “deployed in production”, and I make no claim about live traffic.

1.3 Problem Statement

Small businesses record a lot of order data, but they mostly use it for billing, stock updates and past reports. The records describe real buying relationships, yet nothing turns them into help at the moment of sale. A good waiter or cashier often knows that a certain main dish goes with a drink or a dessert. Self-service QR ordering removes that human suggestion. If the POS only shows generic popular items, it ignores what the customer has already chosen, and the offers can be repetitive or irrelevant.

A second problem is that recommendation methods built for large online platforms usually assume detailed user profiles, very large interaction histories or heavy computing infrastructure. Those assumptions do not fit a micro enterprise. A single branch may have many anonymous or occasional customers, and nobody should have to create an account just to get a sensible suggestion. The logic must also be explainable to non-technical staff, because owners need to understand and trust whatever is influencing their checkout.

The research problem is therefore this: how can historical POS transactions be turned into fast, explainable, stock-aware and cart-aware product recommendations that are practical for micro-enterprise restaurant operations and can be built into an existing ordering workflow? My answer is a lightweight hybrid recommender that uses basket mining, item similarity, personal history, category-level behaviour and popularity, without a conventional login-based customer profile.

1.4 Research Gap

Earlier work has shown that association-rule mining, collaborative filtering and hybrid recommendation all work for retail data [3], [4], [6]. The gap I address is narrower and concerns deployment. I could not find an approach that runs inside a practical small-business POS workflow, uses branch-specific operational data, stays understandable to business users and serves anonymous QR customers in real time. POS products tend to concentrate on processing transactions, while recommender platforms often assume larger datasets, stronger user identification or more infrastructure.

The gap is not simply the lack of a recommendation algorithm. A usable small-business solution has to do several things at once. It must build valid baskets from

existing sales records, remove products that the branch has run out of, avoid suggesting what is already in the cart, work when the customer is not identified, combine strong association rules with weaker similarity and popularity signals, explain its choices in plain language, and deliver the result at a point where the customer can act on it. The Smart Order Recommendation System treats this as one integrated problem, not as an isolated test of market-basket mining.

The component also goes beyond a design that relies on association rules alone. Apriori rules remain a core signal, but they are combined with item-item cosine similarity, customer favourites, category attach rates and overall popularity. Each signal is put on a common scale and weighted before ranking. The aim is better coverage when a candidate is relevant but does not meet the support threshold for a formal rule. A last diversity step prevents one reason type from filling the whole list. Table 1.2 sets out the limitations I observed and how the component responds to each.

Table 1.2: Research gap and the corresponding response of the proposed component

Observed limitation	Why it matters in a micro-enterprise POS	Response of the recommendation system
Generic popularity lists	They ignore the content of the cart and may suggest redundant items.	Cart-aware hybrid scoring with rules, similarity and category context.
Dependence on customer accounts	QR customers may not log in or create profiles.	Optional mobile-number favourites; recommendations still work without it.
Recommendations that ignore availability	A suggested product may be unavailable at the branch.	Stock-aware candidate filtering before scoring.
Black-box logic	Staff may not trust suggestions they cannot understand.	Reason codes converted into short customer-facing explanations.
Single-technique recommenders	A strict support threshold can miss weaker but useful relationships.	Fusion of Apriori, item similarity, category attach rate, favourites and popularity.
Analytics disconnected from checkout	Insights may sit on dashboards and never influence an order.	The suggestion dialog is placed directly before QR order submission.

1.5 Aim and Objectives

The aim of this individual component is to design, implement and evaluate a lightweight hybrid recommendation system that uses historical branch-level POS transactions to produce relevant, explainable and usable product suggestions during

restaurant ordering and checkout. Table 1.3 lists the eight objectives that follow from this aim.

Table 1.3: Research and product objectives of the Smart Order Recommendation System

Objective	Description
O1	To transform completed branch-level orders into market-basket data suitable for recommendation mining, removing duplicates while keeping product co-occurrence.
O2	To identify frequent itemsets and interpretable product association rules using Apriori and the measures support, confidence and lift.
O3	To extend association-rule recommendations with item-item cosine similarity, category attach behaviour, customer favourites and overall popularity in one hybrid ranking model.
O4	To keep recommendations operationally relevant by excluding products already in the cart and products unavailable at the selected branch.
O5	To give each recommendation a reason that QR customers and business staff can understand.
O6	To integrate the workflow with the SmartCost QR ordering page and the staff mobile application without disrupting the existing order and billing process.
O7	To provide repeatable validation using deterministic synthetic order data with planted co-purchase patterns, so the implementation can be checked against known relationships.
O8	To identify the limitations of the current implementation and define measurable future evaluation methods, including top-k offline evaluation and recommendation-uptake measurement.

1.6 Research Questions or Hypotheses

I framed the work as research questions (RQ) and not as statistical hypotheses. The reason is that the current evidence comes from engineering validation and deterministic synthetic data. A full offline top-k evaluation and a controlled business-impact experiment have not been carried out, so a formal hypothesis test would claim more than the evidence supports. Table 1.4 gives the six questions.

Table 1.4: Research questions guiding the individual component

Research question	Question
RQ1	How can historical POS order baskets be converted into product relationships that are interpretable and light enough for a micro-enterprise recommendation workflow?
RQ2	How can association rules, item-item similarity, category attach behaviour, customer favourites and popularity be combined so that coverage is wider than with association rules alone?
RQ3	How can branch stock, current-cart contents and missing cart categories be built in so that recommendations remain valid and relevant to the situation?
RQ4	How can recommendation explanations be generated in plain language while still reflecting the analytical reason for each suggestion?
RQ5	How can the engine be integrated into QR ordering and staff insight screens without changing the basic order-processing and billing workflow?
RQ6	How can the recommendation logic be validated reproducibly when real acceptance labels are not yet stored by the production workflow?

1.7 Scope, Limitations and Assumptions

I kept the scope to what can be derived from POS baskets and a little customer context. Recommendations are calculated for each shop and branch from completed orders in a 365-day lookback period. An order is treated as a set of distinct product identifiers (IDs), and quantities are ignored because Apriori only asks whether an item was present in a basket. A candidate must be an orderable product with a positive price, available at the branch and not already in the cart. The customer QR module runs in the development and testing environment and supports the interaction studied here. Public production hosting is outside the current deployment scope.

The model needs at least ten completed baskets before it counts as ready. Apriori uses a default minimum support count of three, antecedents hold at most two products, the item-item component keeps the top eight similar neighbours, and the engine keeps a bounded set of association rules. The default list length is ten, and the public endpoint limits any requested length to a safe range. Customer favourites are optional and are taken from the products most often bought by a supplied mobile number at the same branch.

Several limitations are known. First, the model is mined again on every request instead of being precomputed and cached, so processing cost will rise with transaction volume.

Second, accepted recommendations are not stored, so true uptake, extra revenue and causal sales impact cannot yet be calculated from production records. Third, no offline hold-out measure such as Precision@K or Hit Rate@K exists in the current code path. Fourth, the five signal weights are design parameters and were not learned from acceptance data. Fifth, products with fewer than the minimum number of orders suffer a cold-start effect, and the engine does not use time of day, day of week or order type. The work assumes that completed historical orders are representative enough of recurring buying patterns, that product IDs and category assignments are consistent, and that branch stock records show availability correctly. The synthetic dataset is used to verify the implementation. It is not offered as proof of real-world business uplift, and any figure for acceptance, sales growth or revenue improvement would need outcome data that the system does not yet collect. Table 1.5 summarises the boundaries.

Table 1.5: Scope boundaries and assumptions

Scope element	Definition
Included	Branch-specific 365-day baskets; Apriori rules; item-item cosine similarity; favourites; category attach rates; popularity; stock filtering; diversity; customer QR integration; staff insights.
Excluded / not yet implemented	Learned signal weights; contextual time, day and order-type features; scheduled model caching; offline Precision@K and Recall@K evaluation; stored acceptance events; causal revenue-uplift measurement.
Primary operational domain	Restaurant and food-service ordering within SmartCost POS, using branch-specific product and order data.
Identity model	Anonymous QR ordering with an optional mobile number for lightweight repeat-customer favourites; no login for the customer endpoint.
Validation scope	Deterministic synthetic transactions with planted buy-together patterns, plus software and integration testing of the implemented workflow.

1.8 Individual Contribution

My contribution covers the recommendation pipeline from data preparation to the screens that people use. On the data side, the component builds transaction baskets from completed order history, constructs an orderable and stock-aware candidate catalogue, derives favourites from optional mobile-number history, and identifies the categories already present in the cart. On the analytical side, it applies Apriori frequent-itemset mining, derives association rules and computes support, confidence

and lift. It also builds an item-item sparse matrix for cosine similarity, calculates category attach rates and product popularity, and fuses the signals into one normalised ranking score.

I also treated explainability and diversity as requirements in their own right. For each candidate, the system finds the strongest contributing signal and maps it to a reason the user can read. If the top of the list would carry only one type of reason, the last slot can be replaced by the strongest candidate with a different explanation. This keeps the suggestions more varied while leaving the order deterministic when scores are equal.

At the integration level, the component exposes a public branch recommendation endpoint for QR customers and a protected staff insights endpoint. The public endpoint accepts cart items, an optional mobile number and a requested limit. The staff endpoint uses the authenticated shop and branch to return model statistics, rules and category attach information. The Python endpoint is hidden behind the Node.js application layer, so the interfaces look the same as the rest of SmartCost even though the engine is written in Python.

For reproducibility, I wrote a deterministic seed-data process that can generate 12,000 synthetic orders with planted relationships: main dish to beverage, Rice & Curry to a seafood side, main dish to dessert, and grilled or devilled seafood to Fresh Lime Juice. Because the planted relationships are known in advance, the data gives a repeatable way to check whether the algorithm rediscovers the expected rules with meaningful lift. It verifies the implementation. It does not stand in for real customer behaviour. Table 1.6 summarises the contribution areas.

Table 1.6: Summary of the individual contribution

Contribution area	Individual contribution
Data preparation	Branch-specific baskets, catalogue filtering, stock availability, favourites and cart-category context.
Algorithmic design	Apriori with association rules, item-item cosine similarity, category attach rates, popularity and five-signal score fusion.
Recommendation quality controls	Cart exclusion, stock exclusion, normalised signals, strongest-reason selection, deterministic tie-breaking and the diversity rule.
Explainability	Readable reasons linked to the dominant analytical signal of each suggested product.

Contribution area	Individual contribution
Customer integration	Recommendation dialog in QR ordering before the final review and send step.
Staff integration	Mobile insights screen with model statistics, buy-together rules and category attach rates.
Reproducibility	Deterministic synthetic order generator with planted ground-truth co-purchase patterns.
Research documentation	A clear statement of current limitations and future evaluation requirements, without unsupported business-impact claims.

1.9 Thesis Structure

The remaining chapters move from research foundations to design, implementation, validation and practical consequences. They follow the approved thesis contents and keep the recommendation component tied to the complete SmartCost solution. Table 1.7 gives the purpose of each part.

Table 1.7: Organisation of the remaining thesis

Thesis section	Purpose
Chapter 2 – Literature Review and Related Work	Reviews recommender-system foundations, market-basket analysis, association rules, collaborative filtering, hybrid methods, explainability, existing POS solutions and the research gap for micro-enterprise deployment.
Chapter 3 – Research Methodology	Defines the research design, study materials, procedure, measures, analysis, ethics, privacy and security considerations, and the artefacts that make the work reproducible.
Chapter 4 – Product and Solution Design	Specifies the requirements, component architecture, design alternatives, detailed algorithm and application programming interface (API) design, and professional-practice considerations.
Chapter 5 – Implementation, Integration and Testing	Documents the environment, code structure, Node.js orchestration, the Python/FastAPI recommender, the Next.js and React Native integrations, verification tests and deployment.
Chapter 6 – Results and Validation	Presents the results the available evidence supports. Where a metric is not implemented, the chapter says so and does not invent a result.
Chapter 7 – Discussion	Interprets the findings, compares the approach with the literature and baseline techniques, explains the research and product contribution, and covers threats to validity and implications.
Chapter 8 – Commercialisation and Sustainability	Covers the value proposition, market position, business model, scalability, intellectual property, compliance and Sustainable Development Goal alignment.
Chapter 9 – Conclusion and Future Directions	Summarises the study, maps outcomes to objectives, states the individual contributions and lists future work.
References, Glossary and Appendices	Provide the bibliography, term definitions, extended evidence, research artefacts and the academic-integrity and artificial intelligence (AI) use disclosure.

In short, this chapter has set out why the component exists and how far it goes. The research problem is more than finding frequent item pairs. It is to build a recommendation capability that is operationally valid, explainable, lightweight, branch-aware and part of a real POS ordering workflow. The next chapter reviews the theory and earlier work needed to position that contribution.

2 LITERATURE REVIEW AND RELATED WORK

2.1 Review Method and Scope

This chapter reviews the work that shaped the Smart Order Recommendation System. I limited the review to methods that matter for recommending products from transaction data in small and micro businesses. The main themes are POS data as evidence of behaviour, market basket analysis, association rule mining, item-based recommendation, hybrid strategies, explainability, and what real-time checkout integration demands. I also looked at the practical constraints of small-business systems: low computing cost, little reliance on rich user profiles and logic that people can follow.

The sources fall into three groups. For data-mining concepts I relied on Han, Kamber and Pei [1], Witten et al. [2], Hand, Mannila and Smyth [7] and Aggarwal [8]. Association rule mining rests on the Apriori work of Agrawal and Srikant [3]. For recommender systems I used Resnick and Varian [9], Schafer, Konstan and Riedl [10], Adomavicius and Tuzhilin [6], Sarwar et al. [4], Deshpande and Karypis [5] and the Recommender Systems Handbook [11]. These works explain why the techniques are sound. They are used to interpret what SmartCost does, not to replace it with a generic design.

The review is narrower than a survey of artificial intelligence in retail. Deep neural recommenders, large language model recommenders, image-based recommenders, advertising systems and large streaming architectures are not baselines here, because the implemented system uses none of them. I concentrated on the techniques that can be compared with what was built: Apriori association rules, item-item cosine similarity, category attach behaviour, popularity, light personal history, stock filtering, score fusion and diversity.

I also keep two kinds of evidence apart. Academic literature shows why market basket analysis and recommender methods are valid. The SmartCost technical documentation shows how this particular system applies them: the 365-day branch window, the ten-basket minimum, the stock filter and the five-signal score. The first tells us whether an

idea is reasonable. The second tells us what was actually built. Table 2.1 links each theme to the system.

Table 2.1: Scope of the literature review

Theme	Purpose in this thesis	Relation to the implemented system
POS and transactional analytics	Shows the value of historical sales data for operational decision support.	Completed order baskets are the main behavioural input.
Association rule mining	Explains frequent itemsets and co-purchase rules.	Apriori produces rules evaluated with support, confidence and lift.
Item-based recommendation	Explains similarity between products based on co-occurrence.	Item-item cosine similarity gives a weaker but broader complementary signal.
Hybrid recommendation	Looks at combining several signals to offset the weaknesses of one method.	Rules, similarity, favourites, category attach rates and popularity are fused into one score.
Explainability	Addresses trust and interpretability in operational recommendations.	Each suggestion is returned with a readable reason.
Real-time POS integration	Focuses on responsiveness and fit with the workflow.	Recommendations are requested during QR order review and also shown in staff insights.

2.2 Domain and Theoretical Foundations

2.2.1 POS transaction data as behavioural evidence

POS systems were first built to record sales, print bills and control stock. The same records also hold a history of what customers buy together. In data-mining terms, each completed order is a transaction, or basket, containing a set of items. When many baskets pile up, repeated co-purchase structures can be found and turned into decision-support knowledge [1], [2]. That shift from recording to analysing is central here, because the system learns from orders that were already completed, not from explicit ratings or product links written by hand.

SmartCost treats every completed order as a set of distinct product IDs. Quantity is ignored for association analysis, because what matters is whether a product appeared in the basket and not how many units were bought. This matches standard market-basket practice, where binary occurrence is enough for counting and rule generation. The system uses completed orders from the previous 365 days for the relevant shop

and branch. A longer window gives more observations to learn from, and the branch scope keeps each location's own behaviour.

2.2.2 Market basket analysis and frequent itemsets

Market basket analysis is the retail use of association analysis. The goal is to find items that keep turning up together in transactions. Agrawal and Srikant [3] described efficient ways to mine frequent itemsets and to generate association rules from them. The underlying idea is that a combination which occurs often enough is probably a real buying relationship and not coincidence. In a POS setting that supports cross-selling: a product already in the cart is evidence for suggesting another product that is usually bought with it.

Apriori saves effort through downward closure. If an itemset is frequent, all of its subsets must be frequent, and if a subset is infrequent, every larger set containing it can be dropped without being counted. SmartCost applies this to itemsets of up to three products, with a default minimum support count of three. The model finds frequent single products first, then frequent pairs, and goes on to triples only when the smaller sets survive. This keeps mining feasible at small-business transaction volumes and leaves the patterns easy to read.

2.2.3 Association rules: support, confidence and lift

Frequent itemsets become useful recommendations once they are turned into directional rules of the form $A \rightarrow B$, where A is the antecedent and B the consequent. In the implemented system the antecedent holds one or two products and the consequent is a single product, because the consequent is the item that can be recommended. Rule quality is judged with support, confidence and lift [3], [8]. Table 2.2 explains each measure and what it does in the recommender.

Support tells us how often A and B appear together across all N baskets: $\text{support}(A \rightarrow B) = \text{count}(A \cup B) / N$. Confidence is the conditional likelihood of seeing B when A is present: $\text{confidence}(A \rightarrow B) = \text{count}(A \cup B) / \text{count}(A)$. Lift compares that likelihood with how popular B is overall: $\text{lift}(A \rightarrow B) = \text{confidence}(A \rightarrow B) / \text{support}(B)$. A lift above one means B appears with A more often than its general popularity would predict. The implementation discards every rule with lift of one or

less, so a neutral or negative association can never become the strongest co-purchase signal.

Table 2.2: Interpretation of association-rule measures

Measure	Interpretation	Use in recommendation
Support	How common the full item combination is across all orders.	Stops recommendations from resting on extremely rare coincidences.
Confidence	Probability of the consequent when the antecedent is present.	Shows how reliably the cart condition leads to the recommended product.
Lift	Strength of the rule relative to the consequent’s baseline popularity.	Separates a genuine relationship from a product that is simply popular overall.

2.2.4 Item-item collaborative filtering and cosine similarity

Further evidence for item-based methods comes from Linden et al. [24], who describe item-to-item recommendation at e-commerce scale, and Karypis [25], who evaluates item-based top-N ranking. These works support the selection of item relationships as an alternative to explicit customer profiles; they do not establish that the particular SmartCost weights or product rankings are optimal.

Association rules work well when a combination clears the support threshold, but they can miss weaker relationships that are still useful. Item-based collaborative filtering offers another view by comparing how products co-occur across transactions. Sarwar et al. [4] and Deshpande and Karypis [5] developed item-oriented methods that use similarity between items to produce top-N recommendations. In a POS setting this has a practical advantage: suggestions can be based on what is in the current basket and need no long history for the current user.

The system builds a binary basket-by-item matrix and computes cosine similarity between product columns. Two products are highly similar when they tend to appear in the same baskets relative to how often each appears alone. For products i and j , the measure equals $|B_i \cap B_j|$ divided by the geometric mean of $|B_i|$ and $|B_j|$. Each item keeps a limited number of neighbours, eight by default. The signal is weighted below association rules because it supplies a supporting “goes well with” link and is not the main mechanism.

2.2.5 Hybrid recommendation and score fusion

Burke [19] distinguishes several ways of combining recommendation methods, including weighted hybrids. The SmartCost weighted fusion is consistent with that design family, although its five weights were selected as engineering parameters rather than estimated from feedback. Koren et al. [29] and He et al. [35] describe more data-intensive latent-factor and neural alternatives; these provide useful comparators for future experiments rather than methods implemented in this thesis.

A recurring finding in recommender research is that each technique has typical weaknesses. Popularity methods are dependable but not personal. Collaborative filtering can capture similarity but struggles when interaction histories are sparse. Rule-based methods are explainable but need enough repeated co-occurrence. Personal history is simply unavailable for anonymous users [6], [11]. Hybrid systems combine signals so that one source of evidence can make up for another that is weak.

The Smart Order Recommendation System follows that idea. Five signals can contribute to a candidate: frequently bought together, personal favourite, similar taste, popular in a missing category, and popular overall. Each is first expressed on a scale from zero to one and then weighted. The weights are 1.00 for association-rule strength, 0.85 for personal favourites, 0.70 for item similarity, 0.60 for category attach behaviour and 0.35 for overall popularity. The total score is the weighted sum of the signals that apply. Association evidence stays dominant, but the system does not depend on one analytical method.

The rule signal needs extra care because lift has no upper bound. The implementation transforms lift as $\text{lift}/(1 + \text{lift})$ and averages the result with confidence. This produces a bounded rule-strength value that can sit beside cosine similarity, attach rate and popularity without letting a very large lift swamp the other signals. Only the strongest matching rule is used for a candidate. That prevents a product backed by many weak rules from outranking one backed by a single clearly stronger rule.

2.2.6 Explainability, diversity and availability constraints

Recommendation quality extends beyond predicting an item accurately. Ziegler et al. [21] investigate diversification of recommendation lists, while Kaminskas and Bridge [27] review diversity, novelty, serendipity and coverage.

SmartCost’s simple reason-diversity rule is motivated by these concerns but has not been evaluated against their metrics. Research on local and recommendation-specific explanations [32], [33] also shows why a reason should be both comprehensible and faithful to its underlying evidence; the present explanation labels require user validation.

Interpretability matters in small-business tools because the outputs are read by non-specialist staff and by customers at checkout. Association rules already have a natural explanation: a suggestion is “often ordered with” something in the cart. The system extends this idea to every scoring signal. A candidate can be labelled “You ordered this before”, “Goes well with” a cart item, “Most loved” in a category, or “Popular right now”. The reason shown is the signal that contributes most to the final score, so each recommendation can be traced to its strongest evidence.

Two practical controls are not always present in abstract recommender formulations. First, candidates are stock-aware: products that are unavailable at the branch or already in the cart are excluded. Second, a diversity rule applies when the best-ranked list would carry the same reason throughout. In that case the final position can be taken by the strongest candidate with a different reason. This adds variety without a separate diversity-learning model.

2.3 Related Research

The sparse-feedback problem has established precedents. Schein et al. [28] examine cold-start recommendation, whereas Rendle et al. [22] develop ranking from implicit interactions. These studies motivate future alternatives when transaction histories grow, but the current component does not train a Bayesian ranking model or solve cold start for newly introduced products. Session-based recurrent recommendation [34] represents a further alternative if SmartCost

later captures ordered interaction sequences rather than only completed baskets.

Research on recommender systems has grown from early information-filtering ideas into a broad family of methods for personalised ranking. Resnick and Varian [9] described recommender systems as a way of helping people choose from a larger set of options. In electronic commerce, Schafer, Konstan and Riedl [10] presented recommendation as a practical tool for merchandising, cross-selling and a better shopping experience. This is relevant to POS recommendation, where the aim is less to predict preference in the abstract and more to make a timely suggestion that can change what ends up in a transaction.

Collaborative filtering became a leading direction because it can infer recommendations from collective behaviour. Sarwar et al. [4] demonstrated item-based collaborative filtering, and Deshpande and Karypis [5] developed item-based top-N algorithms. These approaches work best when the interaction matrix is reasonably dense. In micro-enterprises that is often not the case: customer identity may be incomplete, repeat visits are hard to link, and many orders are anonymous. For that reason the system uses item-item similarity as one component and does not rely on a customer-item model.

Association rules form another strand. Agrawal and Srikant [3] showed how frequent itemsets can be mined efficiently and turned into interpretable rules, which fits the search for complementary products in retail baskets. The group's earlier SmartCost research paper took this view as its main mechanism for real-time checkout suggestions and stressed support, confidence and lift. The current implementation keeps that foundation and adds item-item similarity, favourites, category attach rates, popularity, weighted fusion and a diversity rule. It can be seen as a step from a pure market-basket recommender towards a lightweight hybrid.

Hybrid recommendation, as surveyed by Adomavicius and Tuzhilin [6] and by Ricci, Rokach and Shapira [11], supports combining different kinds of evidence. This helps most where one method cannot cover enough ground. In the SmartCost setting, rules may be missing for new or rarely paired items, item similarity gives neighbourhood

evidence, personal history helps returning customers, category attach rates cover a missing part of the meal, and popularity is a weak fallback. The combination follows the general rationale for hybrids, simplified for resource-constrained POS use.

A further strand concerns data-driven decision making in business. Chen, Mao and Liu [12] and McAfee and Brynjolfsson [13] describe how large volumes of operational data can improve decisions. Their discussion is usually about large organisations, but the principle holds at small scale: transaction histories are an underused source of knowledge. SmartCost applies the idea inside the checkout flow and does not leave analytics to retrospective dashboards, and it also shows the mined patterns to staff as buy-together rules and category attach rates.

Four conclusions follow from this body of work. Transaction baskets are a suitable representation for learning co-purchase patterns. Association rules give interpretable and efficient evidence. Item-based similarity adds coverage that rules lack. Hybridisation can combine several weak and strong signals into a sturdier ranking. What remains open is how to fit these methods into a practical POS architecture for micro-enterprises that have neither extensive customer profiles nor enterprise-scale infrastructure. The next section compares existing approaches with that question in mind.

2.4 Existing Systems, Products and Techniques

Existing recommendation solutions differ in the evidence they use and in where the recommending happens. Traditional POS systems mainly handle billing, inventory and transaction history. A simple extension is popularity-based recommendation, which promotes frequently sold products regardless of the current basket. More advanced systems use association rules, collaborative filtering or hybrids. Enterprise e-commerce recommenders may add content features, user embeddings, deep learning and large-scale online experiments. These options differ a great deal in data needs, interpretability, infrastructure cost and suitability for a small business. Table 2.3 compares them.

Table 2.3: Comparison of recommendation techniques relevant to small-business POS environments

Technique	Primary evidence	Strengths	Limitations for this context
Manual cashier recommendation	Staff experience and memory	No technical infrastructure; can use human context.	Inconsistent, hard to scale and not systematically data-driven.
Popularity ranking	Overall sales frequency	Simple, dependable and inexpensive.	Ignores the current basket and gives identical suggestions to different customers.
Association-rule recommendation	Frequent co-purchase patterns	Explainable, transaction-oriented and suited to market baskets.	Needs repeated co-occurrence; limited coverage for sparse or new items.
Item-based collaborative filtering	Similarity among products from interactions	Can find related items beyond strict frequent rules.	Depends on enough interaction data and on the similarity computation.
Customer collaborative filtering	Repeated user-item histories	Supports stronger personalisation.	Difficult when QR customers are anonymous or identity is incomplete.
Enterprise hybrid recommender	Behavioural, content and model signals	High flexibility and potentially broad coverage.	Often complex, expensive and demanding to operate for a micro-enterprise.
Smart Order Recommendation System	Rules, item similarity, favourites, category attach and popularity	Lightweight hybrid; explainable, cart-aware, stock-aware and branch-specific.	Weights are hand-tuned; cold start and production-scale optimisation remain open.

The comparison shows why a popularity-only method does not meet the SmartCost need. Popularity can pick out a drink or dessert that sells well in general, but it cannot say whether that product suits what the customer has already chosen. The recommender therefore gives popularity the lowest weight and uses it mostly as a weak ranking signal. Association rules carry the highest weight because they express evidence that depends directly on the basket, while cosine similarity and category attach behaviour widen the pool of candidates.

A purely collaborative approach is also a poor fit for QR ordering, because customers do not log in. The only personal signal is an optional mobile number. When one is given, the system retrieves up to ten products that this number has ordered most often at the branch. When it is missing or invalid, the request still succeeds and personalisation is simply skipped. That is the main difference from recommenders that need stable user accounts before they can rank anything usefully.

The group’s earlier SmartCost research paper describes a lightweight association-rule architecture with React Native, Node.js, FastAPI and MongoDB, with real-time suggestions built into checkout. That paper gives useful motivation and an evaluation framing. The current implementation, however, is more specific and more hybrid. The customer interface is a Next.js QR ordering page, the staff interface is an Expo/React Native application, and the Python service performs Apriori mining, item-item cosine similarity, category attach calculation, score fusion and diversity handling. For this reason I treat the current implementation as the authoritative description of the product and the earlier paper as supporting context. Table 2.4 shows how requirements drawn from the literature map onto the implementation.

Table 2.4: Literature-derived design requirements mapped to the Smart Order Recommendation System

Requirement	Rationale from reviewed work	Implementation response
Context-sensitive recommendation	Recommendations should respond to what the user is selecting now.	Cart items are included in each request and activate matching rules and similarity signals.
Explainability	Interpretable recommendations improve understanding and trust.	Each result carries a reason code that is translated into a plain-language explanation.
Low operational overhead	Micro-enterprises have limited infrastructure and technical resources.	A lightweight Node.js and FastAPI service architecture is used instead of a large model stack.
Coverage beyond one method	Single-method recommenders can fail under sparse or cold conditions.	Five signals are fused, with popularity and category attach rates widening coverage.
Inventory validity	A checkout recommendation must be orderable now.	Branch stock is checked and unavailable products are removed from candidates.
Anonymous-customer support	QR checkout should not require an account.	Personal history is optional; recommendation works without a phone number.
Operational integration	Recommendation value depends on appearing in the purchasing workflow.	Suggestions are requested at the review stage and can be added directly to the cart.

2.5 Gap Synthesis and Conceptual Framework

The literature shows that the core algorithms for retail recommendation are well established. Association rules identify frequent co-purchases, item-based collaborative filtering represents product similarity, and hybrid methods combine several sources of

evidence. The gap this thesis addresses is therefore not the invention of association rules or cosine similarity. It lies in configuring and integrating these known techniques into a lightweight, explainable, stock-aware and branch-specific component that works inside a micro-enterprise POS ordering workflow.

Several practical mismatches appear between generic recommender research and the SmartCost setting. Many recommenders assume stable user profiles, whereas QR customers can be anonymous. Some techniques optimise prediction accuracy without asking whether the product is available. Enterprise architectures can demand engineering capacity out of proportion to a small business. Pure popularity lists ignore the current basket, while pure association-rule systems may cover too little when a rule misses the support threshold. These mismatches motivate the hybrid design, and Table 2.5 lists them with the matching design responses.

Table 2.5: Synthesis of research gaps and design responses

Observed gap	Effect on small-business POS recommendation	Design response
Limited use of transaction data during checkout	Historical data stays retrospective and is not actionable at the moment of sale.	Historical baskets are mined when recommendations are requested during order review.
Dependence on customer identity	Anonymous QR users cannot be served by profile-dependent methods.	Cart and branch data drive the model; mobile-number favourites are optional.
Single-technique coverage limits	Rules may be absent and popularity may be too generic.	Association rules, cosine similarity, category attach rates, favourites and popularity are combined.
Weak explainability in complex models	Cashiers and owners may not understand why an item is suggested.	The dominant scoring signal is converted into a readable reason.
Recommendations that ignore availability	Unavailable products may be shown to customers.	Candidate generation filters branch stock and current cart contents.
Enterprise-scale complexity	High computing and maintenance demands reduce suitability for small businesses.	A modular Node.js/FastAPI architecture with lightweight analytical methods.
Lack of recommendation variety	The top-N output may become repetitive.	A diversity rule brings in a different reason type when the list is uniform.

Figure 2.1 shows the conceptual framework that came out of this synthesis. It also makes the role of each signal clear. Association rules are the strongest evidence of a

purchase relationship that depends on the cart. Item-item similarity extends the model to related products that do not form a strong rule. Personal favourites bring in limited individual history without a login. Category attach rates identify categories that commonly appear in orders and can point to one missing from the basket. Overall popularity is a low-weight global prior. Score fusion joins these signals, and the diversity rule keeps one reason from dominating the list.

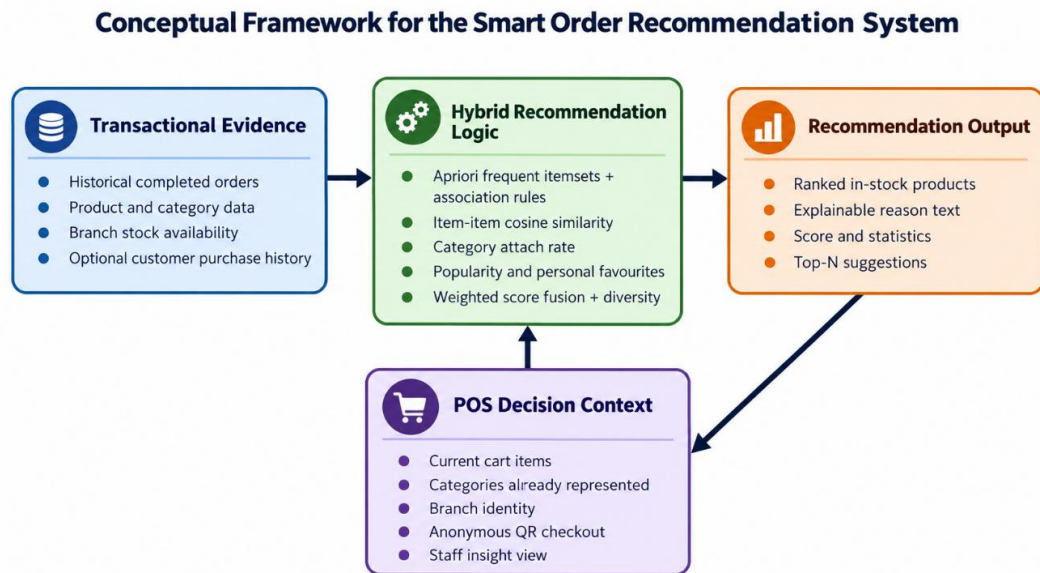


Figure 2.1: Conceptual framework for the Smart Order Recommendation System

From a research point of view, the framework sets out the variables and relationships examined in later chapters. Historical transaction structure affects the quantity and quality of the frequent itemsets and similarity relationships. The current cart determines which learned relationships apply. Candidate availability limits the feasible set. The weighted combination of evidence decides the ranking, and support, confidence, lift, similarity, attach rate and final score serve as interpretable technical measures. The implementation and validation chapters ask whether the framework can be realised within the SmartCost architecture and whether the outputs match the patterns planted in the validation data.

The review therefore supports the design direction and also marks its limits. The contribution is a practical way of combining and integrating known techniques for micro-enterprise POS use. It is not a claim that the underlying algorithms are new. The

next chapter turns this conceptual basis into a research methodology covering data preparation, procedure, evaluation measures, analysis, ethics and reproducibility.

3 RESEARCH METHODOLOGY

This chapter explains how the Smart Order Recommendation System was designed, built and checked inside the SmartCost POS environment. The method is based on the implemented code and on how that code is documented to behave. The central idea is to treat completed POS orders as transaction baskets, derive interpretable item relationships from them, combine several recommendation signals, and deliver ranked suggestions at checkout. The study therefore mixes design-oriented software research with transaction-pattern analysis and controlled validation on reproducible synthetic order data.

The system is not a single association-rule method. It combines Apriori mining with item-item cosine similarity, category attach rates, overall popularity, optional customer favourites, weighted score fusion, stock-aware filtering and a diversity rule. This matters because the group's earlier research paper described a simpler, association-rule-centred method. The current implementation has moved on to a hybrid market-basket design, so I treat it as the main artefact of study and use the earlier description only where it still agrees with the code.

The methodology also separates checking that the implementation behaves correctly from measuring how effective the recommender is. The codebase includes deterministic synthetic data with planted purchasing patterns, which lets the engine be tested against known relationships. It does not yet store accepted recommendations or compute an offline hold-out measure such as Precision@K or Hit Rate. This chapter therefore describes what the available artefacts can validate and does not present unsupported accuracy or business-impact figures as results.

3.1 Research Design

The study follows a design-and-evaluation approach. A software artefact is built to address a practical problem in micro-enterprise POS operations and is then assessed through controlled technical validation. The problem is the loss of contextual cross-selling help during self-service or cashier-assisted checkout. The artefact is the Smart

Order Recommendation System, which turns historical completed orders and live cart context into explainable, branch-specific product suggestions.

At system level the design is a modular pipeline, shown in Figure 3.1. Completed orders are read from the MongoDB transaction history. The Node.js service prepares the order baskets, the orderable product catalogue, branch-level stock state, customer favourites where available, and the categories already in the cart. These inputs go to the Python FastAPI analysis service, which does the pattern mining, similarity computation, signal scoring, ranking and diversity control. The ranked result returns to the operational backend, where product details and reason text are added before it is shown to the customer or to staff.

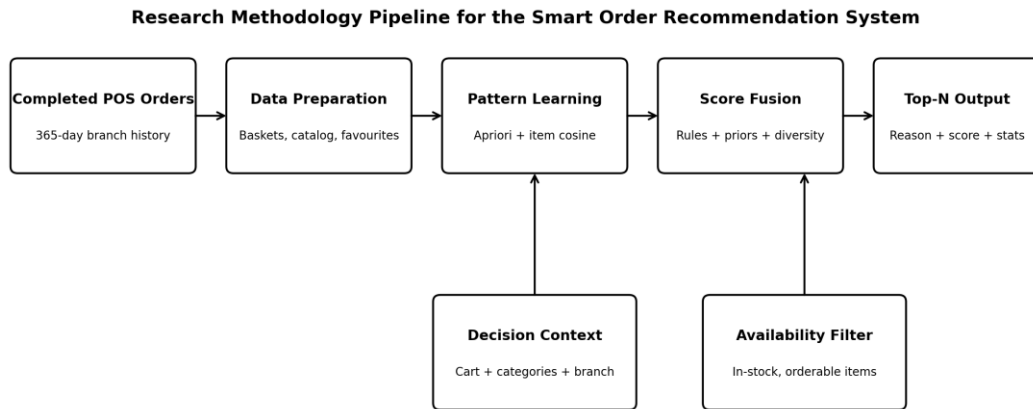


Figure 3.1: Research methodology pipeline for the Smart Order Recommendation System

The design is branch-aware. Recommendation data are scoped by shop and branch identifiers, so patterns learned in one branch are not carried over to another. Completed orders from the most recent 365 days form the evidence for a request. Each order is a set of distinct products and not a quantity-weighted vector, since the Apriori logic asks whether an item occurred in a basket and not how many units were sold.

The design also includes a readiness rule. If fewer than ten baskets are available, the engine reports that the model is not ready and returns no recommendations. This is a safeguard against treating a handful of transactions as evidence of a pattern. Once a branch passes the threshold, the defaults are a minimum support count of three,

antecedents limited to two products, eight nearest neighbours in the similarity component, and at most two hundred association rules kept after filtering and ranking. Table 3.1 summarises these design choices.

Table 3.1: Research design summary

Design element	Implemented choice	Methodological purpose
Research strategy	Design-oriented software research with controlled validation	Build and assess a working recommendation artefact within the SmartCost POS context.
Unit of analysis	Completed order basket	Represents the co-purchase context used by market-basket mining.
Temporal scope	Most recent 365 days	Gives enough transaction volume while keeping recommendations relevant to the branch.
Context scope	shopId + branchId	Prevents cross-branch contamination of purchasing patterns.
Model readiness	Minimum of 10 baskets	Avoids deriving patterns from extremely sparse evidence.
Recommendation approach	Hybrid market-basket recommender	Combines complementary signals and does not depend on one technique.
Validation strategy	Deterministic synthetic data with planted patterns	Allows reproducible checking against known expected relationships.

3.2 Data, Participants or Study Materials

The main study material is the transactional order data produced by the SmartCost POS environment, together with product, category, stock, branch and optional customer-history records. The implementation reads the History collection as its source of completed order baskets. Only orders linked to the active shop and branch, and recorded inside the configured 365-day lookback, are considered. Inside each qualifying order, duplicate product IDs are removed so that a product counts once whatever the quantity.

Three groups of data are prepared in parallel for every recommendation request. The first is the set of historical order baskets. The second is the orderable catalogue, which holds products with a valid positive price and, where inventory tracking is on, the branch-level stock state. The third is optional customer purchase history. For an anonymous QR customer, the mobile number entered during ordering serves as a light identifier for finding the ten products most often bought by that number at the branch.

If the number is missing or invalid, the request carries on without personalisation and does not fail.

The current cart is another essential input. Products already in the cart are removed from the candidate set, and the categories they belong to are passed to the analysis service. This is needed because the engine uses category attach rates to spot missing categories. For example, if a main-dish category is already in the cart but no beverage or dessert is, the category signal can favour suitable products from those missing groups.

For reproducible technical validation the codebase includes a synthetic data generator. The documented seed procedure creates 12,000 synthetic orders across 180 days by default. The target branch can be configured and defaults to shop SI000001 and branch B00001. The generator uses a fixed random seed of 20260827, so repeated runs produce the same order sequence and therefore the same ground-truth structure. Service-hour proportions, basket openers, random noise and planted product relationships are all controlled explicitly.

The synthetic data contains planted relationships that act as known reference patterns. The configuration sets a 70% attach probability from Kottu or Fried Rice to a beverage, 45% from Rice and Curry to a seafood side, 35% from any main dish to a dessert, and 60% from grilled or devilled seafood to Fresh Lime Juice. Most of each category-level relationship is directed to a signature partner product, which makes the item-level association strong enough for Apriori to find.

The earlier research paper also describes a dataset of 15,420 bakery and café transactions over twelve months. Since this thesis documents the current codebase, I treat that dataset as background and do not substitute it for the reproducible validation data. Where the two sources differ, the implementation documentation and its executable seed procedure define what is reported here. Table 3.2 lists the study materials.

Table 3.2: Data and study materials

Data / material	Source in the current system	Use in the methodology
Completed order baskets	History orders for the active branch	Frequent itemsets, association rules, item similarity, popularity and category rates.
Product catalogue	Product records with a valid price	Defines the universe of recommendation candidates.
Branch stock	BranchStock for inventory-tracked products	Excludes unavailable products before results are returned to users.
Current cart	Customer or staff checkout state	Matches rule antecedents and similar-item neighbours.
Cart categories	Category IDs represented in the current cart	Supports category-gap recommendations.
Customer favourites	Top 10 products linked to an optional mobile number	Adds light personalisation without a login.
Synthetic validation orders	seedRecommendationData.js	Provides deterministic orders with known planted co-purchase relationships.

3.3 Experimental or Study Procedure

The study procedure follows the runtime pipeline step by step, so validation exercises the same logic that produces live recommendations. First the active shop and branch context is established. The backend then retrieves completed orders for the lookback period, builds distinct-item baskets, loads the orderable catalogue, checks branch stock where inventory tracking applies, derives optional customer favourites and records the categories already in the cart.

The prepared data are serialised into the internal recommendation request. That request holds the historical transactions, the cart item IDs, the cart category IDs, the candidate products with their category information, optional favourite IDs, the requested number of recommendations and the model-tuning parameters. The defaults are a limit of ten, a minimum support count of three, a maximum antecedent size of two and eight item neighbours. The public endpoint clamps the requested limit to the supported range of one to twelve. Table 3.3 lists the core parameters.

Table 3.3: Core recommendation parameters used in the study procedure

Parameter	Default / rule	Role
Lookback period	365 days	Defines the historical evidence window for each branch.
Model-ready threshold	10 baskets	Prevents recommendations from extremely sparse histories.
Minimum support count	3 baskets	Controls frequent-itemset eligibility.
Maximum antecedent size	2 products	Keeps rules interpretable and tractable.
Top item neighbours	8	Sets the size of the cosine-similarity neighbourhood.
Maximum rules	200	Caps the number of association rules kept.
Default recommendation count	10	Typical number of ranked suggestions returned.
Public limit clamp	1 to 12	Prevents unsupported output sizes.

The first analytical stage is Apriori frequent-itemset mining. Single products meeting the minimum support count are found first. The survivors are combined into pairs, and surviving pairs are extended to triples. Downward-closure pruning avoids evaluating larger candidates when their required subsets are not frequent. Itemsets larger than three are not needed, because rule antecedents hold one or two products and the recommended consequent is a single product.

Association rules are then generated from the frequent itemsets in the form $A \rightarrow B$, where A has one or two products and B is the single product that may be recommended. Rules are evaluated with support, confidence and lift, and any with lift of one or less are removed because they do not show that B becomes more likely when A is present. The remaining rules are ordered by lift, then confidence, then support, and cut off at the configured maximum.

The second learned signal is item-item collaborative filtering based on co-occurrence. A binary basket-by-item sparse matrix is formed and cosine similarity is calculated between item columns, keeping the eight most similar neighbours of each item by default. This signal complements the rules because it can capture weaker co-occurrence that fails the minimum support requirement for rule generation.

The system also calculates category attach rates and item popularity. A category attach rate is the share of baskets containing at least one product from that category, and the

most popular products in each category are kept for category-gap suggestions. Item popularity comes from the single-item support of each product and is the weakest of the five scoring inputs, used as a fallback and as a tie-breaker.

Candidate scoring combines five signals: frequently bought together, personal favourite, similar taste, popular in an unrepresented category, and popular overall, with weights of 1.00, 0.85, 0.70, 0.60 and 0.35. Because lift is unbounded while the other signals fall between zero and one, the lift contribution is compressed as lift divided by one plus lift and averaged with confidence to give the rule strength. This stops large lift values from dominating the fused ranking purely because of their scale.

A candidate's final score is the sum of its weighted signal contributions, and the reason shown to the user comes from the signal that contributes the most. Candidates are sorted by score, with the product ID as a deterministic tie-breaker. The diversity rule then checks whether every recommendation in the selected list has the same reason. If so, the final slot goes to the highest-scoring candidate with a different reason, so that one source of evidence cannot crowd out every other mode.

After ranking, the Node.js backend adds the product name, amount, image, category and stock information and converts reason codes into readable explanations such as "Often ordered with <cart item>", "Goes well with <cart item>", "Most loved <category>", "Popular right now" or "You ordered this before". The results are used in two places. In the customer flow, the "Add something else?" dialog appears before the order is submitted. In the staff flow, the model is called with an empty cart to expose branch-level rules, category attach rates and model statistics.

3.4 Measures and Evaluation Metrics

A completed engineering walkthrough should not be confused with measured recommendation relevance. Herlocker et al. [20] outline the importance of matching evaluation measures to the user task. McNee et al. [26] caution that accuracy alone does not establish usefulness. Accordingly, later evaluation should report offline Precision@K, Recall@K and coverage alongside user-

facing diversity and recommendation uptake; none of these unimplemented metrics is claimed here as a measured result.

Evaluation is divided into rule-quality measures, descriptive model measures, operational validity checks and reproducibility checks. The split is needed because the system exposes several technically meaningful statistics but does not yet run a complete offline recommendation-accuracy experiment. I report the measures the system actually produces and mark the unsupported ones as future evaluation work.

Support measures how often an item combination appears across all baskets (Equation 3.1). Confidence measures how often the consequent B occurs when the antecedent A occurs (Equation 3.2). Lift compares that conditional likelihood with the baseline popularity of B (Equation 3.3), and a lift above one means B is more likely in baskets containing A than in a random basket. Item-item similarity is the cosine measure in Equation 3.4. The rule strength used in scoring is given by Equation 3.5, and the final score of a candidate by Equation 3.6, where w_k is the weight and c_k the normalised contribution of signal k.

$$\text{Support}(A \rightarrow B) = \text{count}(A \cup B) / N$$

$$\text{Confidence}(A \rightarrow B) = \text{count}(A \cup B) / \text{count}(A)$$

$$\text{Lift}(A \rightarrow B) = \text{Confidence}(A \rightarrow B) / \text{Support}(B)$$

$$\text{Cosine}(i, j) = |B_i \cap B_j| / \sqrt{(|B_i| \cdot |B_j|)}$$

$$\text{Rule strength} = 0.5 \times [\text{Lift} / (1 + \text{Lift})] + 0.5 \times \text{Confidence}$$

$$\text{Final score} = \sum (w_k \times c_k)$$

Cosine similarity is calculated from the overlap between the sets of baskets that contain two products. A value near one means the items occur in very similar basket patterns, and a value near zero means they rarely occur together. Category attach rate is the share of all baskets that contain at least one item from a category, and item popularity is the single-item support of a candidate.

The implementation also exposes descriptive model statistics. transactionCount is the number of baskets used for learning, itemCount is how many products pass the minimum support requirement, ruleCount is the number of retained rules with lift above one, pairCount is the number of frequent product pairs, avgBasketSize is the

average number of distinct products per basket, and coverage is the share of items reachable either as rule consequents or through similar-item neighbours.

Synthetic validation adds a ground-truth check. Because the seed data deliberately plants known relationships, the engine can be inspected to see whether the learned rules recover those patterns with meaningful lift. This tests the structure of the mining pipeline. It is not a substitute for a full offline ranking evaluation. The documentation says plainly that no hold-out Precision@K or Hit Rate experiment has been implemented, and that accepted recommendations are not stored by the backend, so recommendation uptake and incremental revenue cannot be calculated. Table 3.4 lists each measure and whether it is available.

Table 3.4: Measures and evaluation metrics

Measure	Definition / interpretation	Current availability
Support	Frequency of a rule's itemset across all baskets	Calculated in the current model.
Confidence	Probability of the consequent given the antecedent	Calculated in the current model.
Lift	Association strength relative to chance	Calculated; rules with lift ≤ 1 are removed.
Cosine similarity	Similarity of item co-occurrence vectors	Calculated for item neighbours.
Category attach rate	Share of baskets containing a category	Calculated and shown in staff insights.
Popularity	Single-item support	Calculated as a weak ranking signal.
Coverage	Share of items reachable by rules or neighbours	Exposed as a model statistic.
Precision@K / Hit Rate	Offline ranking accuracy on held-out items	Not yet implemented in the current codebase.
Recommendation uptake	Accepted suggestions divided by shown suggestions	Not yet measurable because accepted suggestions are not stored.

3.5 Data Analysis

Analysis happens at three levels: transaction transformation, pattern analysis and interpretation of the recommendations. In the transformation step each completed order becomes a basket of distinct product IDs. Empty baskets are removed and repeated occurrences of the same product within an order are collapsed. The resulting

collection is the common representation for the Apriori, popularity, category and item-similarity calculations.

In the pattern-analysis step, frequent-itemset counts generate association rules, and the basket-by-item sparse matrix estimates item-item cosine similarity. Category membership is mapped onto the baskets to give attach rates, and single-item counts become popularity values. These outputs are not used independently at recommendation time. Each candidate product receives contributions from whichever signals apply to it, and these contributions are normalised and fused using the documented weights.

The analysis keeps candidate eligibility separate from candidate ranking. Products already in the cart are removed from the candidate set, and products unavailable at the branch are not offered as valid recommendations. A product can therefore have a strong historical association and still be excluded because the present operational state rules it out. In a POS environment this matters, since recommending something that is out of stock would reduce trust in the system.

Interpretation of the ranked results relies on reason codes and their statistics. For a frequently-bought-together recommendation, the explanation names the relevant cart item. For a similar-taste recommendation, it says the item goes well with a cart item. Category suggestions name a popular missing category, personal favourites refer to prior purchases, and popularity-based suggestions point to general demand. The same output helps with debugging, because each response also carries support, confidence, lift, similarity, popularity, attach rate and final score where they apply.

For research validation, the synthetic dataset is analysed by comparing the mined relationships with the known planted structure. The question is not just whether the engine returns products, but whether the discovered rules and rankings match the relationships built into the generated orders. Because the generator is deterministic, repeated runs can be compared without noise from different random samples.

The analysis makes no causal claim about sales. A recommendation appearing in a top-N list does not show that average order value rose or that an extra purchase was caused. Showing that would require recording which recommendations were displayed

and accepted, then comparing against a control or baseline. The system does not collect that data, so it falls outside the present analysis. Table 3.5 summarises the analysis stages.

Table 3.5: Data-analysis stages

Analysis stage	Input	Output
Basket construction	Completed History orders	Distinct-item transaction baskets
Apriori mining	Transaction baskets	Frequent itemsets and rule counts
Rule analysis	Frequent itemsets	Support, confidence, lift and ranked rules
Item similarity	Binary basket $\times$ item matrix	Top cosine neighbours per item
Category analysis	Baskets plus category map	Category attach rates and top products
Score fusion	Applicable recommendation signals	Deterministic final candidate score and reason
Operational filtering	Catalogue, branch stock and current cart	Orderable, non-duplicate recommendations
Validation analysis	Deterministic planted-pattern dataset	Recovered rules, lift patterns and model statistics

3.6 Ethics, Privacy, Security and Responsible AI

Because optional customer phone numbers and purchase histories can be personal data, safeguards should cover purpose limitation, authorised access, retention and logging. NIST SP 800-53 Rev. 5 [31] provides a recognised catalogue of security and privacy controls for planning such safeguards; citing the standard does not imply that SmartCost has been audited for conformance. Offline evaluation should also account for historical exposure bias: Schnabel et al. [23] explain how observed recommender interactions may lead to biased conclusions if selection effects are ignored.

The system handles transaction records, so it has to respect data minimisation, privacy, access control, transparency and operational safety. The recommendation task does not need sensitive personal profiling. The main evidence is item-level transaction history scoped to a shop and branch. For QR customers, personalisation is optional and uses a mobile number only as a light identifier for looking up earlier product frequency. If the number is unavailable or invalid, recommendation continues without personal favourites.

Using product IDs and basket membership as the main features keeps unnecessary personal data out. Quantities are not needed for Apriori item presence, and the engine requires no demographic attributes, payment-card details or explicit customer profiles. This keeps the analysis focused on buying context and away from sensitive characteristics.

Access control differs between the two interfaces. The customer endpoint is public because QR customers do not log in, but the requested shop and branch are validated and malformed product IDs are rejected. The staff insights endpoint is protected by JSON Web Token (JWT) authentication, and its shop and branch context is taken from the authenticated token. That separation is appropriate because staff insights expose aggregate rule and category information meant for authorised business users and not for anonymous customers.

Responsible behaviour is supported by explanations and availability checks. The system shows plain-language reasons that correspond to the dominant scoring signal and does not present unexplained suggestions. It also filters unavailable items using branch stock and avoids recommending products already in the cart, which reduces the risk of misleading or unusable suggestions.

There are still responsible-AI limitations. The signal weights are set by hand and not learned, so they may reflect design assumptions that no controlled user outcome has tested. The model does not consider time of day, day of week or order type. New products also face cold start, because products with fewer than three orders cannot contribute meaningfully to the learned signals. I document these limits so that the output is not read as universally optimal.

Synthetic data are handled conservatively. The seed tool has a dry-run mode for previewing generated samples and a clean option that deletes only records carrying both the recommendation-system order-ID prefix and the seed-bot user identifier. This lowers the chance of deleting genuine sales records while still allowing repeatable experimental set-up and clean-up. Table 3.6 summarises the controls and what remains to be done.

Table 3.6: Ethics, privacy, security and responsible-AI considerations

Area	Current methodological control	Remaining consideration
Privacy	Optional phone-based favourites; the model mainly uses basket and product data.	Define retention and masking policies for production-scale personalisation.
Security	JWT protection for the staff endpoint; shop and branch validation.	The public customer endpoint still needs normal API hardening and monitoring.
Transparency	Reason text and recommendation statistics are exposed.	Reason wording should stay understandable and not misleading.
Availability safety	Out-of-stock and already-in-cart items are excluded.	Stock synchronisation quality affects how useful recommendations are.
Bias / fairness	No demographic targeting is used.	Popularity and manual weights can still favour established products.
Experimental safety	Dry-run and targeted clean logic for synthetic seed records.	Production and test datasets should stay clearly separated.

3.7 Reproducibility and Research Artefacts

Reproducibility depends on keeping the executable artefacts for the operational backend, the analytical service, the user interfaces and the validation data generator. The recommendation routes, the data-preparation service, the recommendation controller, the Python bridge, the FastAPI router and schemas, the recommendation engine, the customer checkout interface, the staff insight interface and the seed generator together make up the principal research artefacts. Because they follow the implemented request path, another developer can trace a recommendation from incoming cart state to final ranked output.

The implementation documentation names the main artefacts explicitly, and Table 3.7 lists them. The staff route in `recommendationRoutes.js` exposes the protected insights endpoint. The customer route and controller expose the public branch recommendation endpoint. `recommendationDataService.js` prepares baskets, catalogue state and favourites. `recommendationController.js` handles orchestration and reason generation. `recommendationClientViaML.js` and the analysis-service client connect the Node.js backend to FastAPI over the Hypertext Transfer Protocol (HTTP). The Python recommender implements Apriori, association rules, collaborative filtering, scoring and ranking, and the FastAPI router and schemas validate the internal request and response.

The seed script matters most for reproducibility because it removes dependence on one uncontrolled live dataset. It creates a configurable number of orders using a fixed random seed, set service-hour proportions, set basket-opener probabilities, controlled noise and planted relationships. A repeated run with the same parameters should generate the same synthetic evidence, so the same mining behaviour can be examined again.

Reproducibility also requires reporting the model parameters. This thesis records the 365-day lookback, the ten-basket readiness threshold, the support count of three, the maximum antecedent size of two, the top-eight cosine neighbours, the maximum of two hundred rules and the default limit of ten. Stating these values keeps the analytical conditions explicit and avoids leaving the output dependent on undocumented defaults.

The validation workflow is shown in Figure 3.2. Synthetic transactions with known planted relationships are generated first. The recommender then runs through the same pipeline as the application, and the resulting rules and statistics are compared with the known structure. At the same time the limits of the evaluation are kept in the record: no offline hold-out metric is computed, and accepted recommendations are not stored for measuring uptake. Recording both the reproducible checks and the missing metrics stops the methodology from overstating what the artefacts can show.

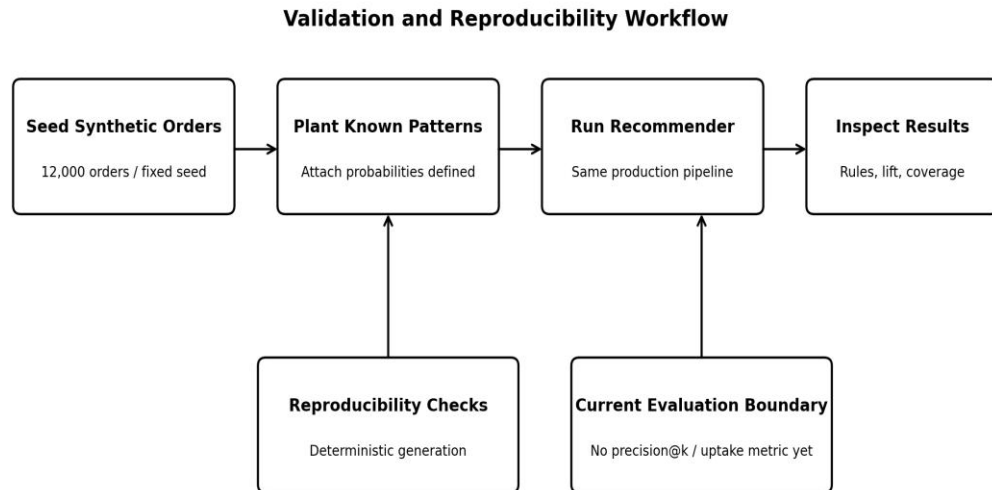


Figure 3.2: Validation and reproducibility workflow

Table 3.7: Reproducible research artefacts

Research artefact	Role
recommendationRoutes.js	Protected staff recommendation insights route.
customerOrderRoutes.js / customerOrderController.js	Public customer recommendation request and validation.
recommendationDataService.js	Prepares baskets, candidates, stock state and favourites.
recommendationController.js	Builds candidates, calls Python and adds user-facing metadata.
recommendationClientViaML.js / analysisServiceClient.js	HTTP bridge between Node.js and FastAPI.
recommender.py	Implements Apriori, association rules, cosine similarity, scoring and diversity.
router.py / schemas.py	FastAPI endpoint and request/response validation.
Next.js QR order page	Displays the recommendation dialog during checkout.
React Native staff recommendations screen	Displays model statistics, rules and category attach rates.
seedRecommendationData.js	Generates deterministic synthetic orders with planted patterns.

4 PRODUCT AND SOLUTION DESIGN

This chapter describes the design of the Smart Order Recommendation System as part of the SmartCost POS platform. The design turns the research objectives into a software structure that can collect transactional evidence, generate recommendations that fit the cart, and show explainable results to QR-ordering customers and to staff. It follows the architecture and decisions documented for the implemented system, and it does not describe features that have not been built.

The design is modular. Operational responsibilities such as request validation, shop and branch context, product retrieval, stock checks and customer-facing wording stay in the Node.js application layer. Recommendation mining, scoring, similarity calculation and ranking are handled by the Python FastAPI analysis service. With this split the recommendation component can change without embedding analytical code in the transaction-processing path.

4.1 Requirements

The requirements come from the problem of missed cross-selling in self-service and staff-assisted POS workflows. The product has to turn existing order history into recommendations that fit the current basket, are clear enough for non-technical users and are safe to show in a live retail setting. I therefore wrote two sets of requirements: functional ones, which say what the system must do, and quality ones, which say how it should behave while doing it.

4.1.1 Functional requirements

The central functional requirement is real-time product suggestions based on the branch's own completed orders. The output is not a generic list of popular products. The system uses the current cart, historical co-purchase patterns, item similarity, category coverage, popularity and optional returning-customer history to decide which available products to suggest. The eleven functional requirements (FR) are listed in Table 4.1.

Table 4.1: Functional requirements of the Smart Order Recommendation System

ID	Requirement	Design interpretation
FR-01	Prepare branch-specific transaction baskets	Use completed orders from the selected shop and branch within the configured lookback period.
FR-02	Generate cart-aware recommendations	Use the items currently in the customer cart as contextual input.
FR-03	Mine frequent itemsets and association rules	Identify recurring co-purchase patterns with Apriori and rule metrics.
FR-04	Calculate item-item similarity	Use cosine similarity to find weaker but useful product relationships.
FR-05	Use category attach rates and popularity	Support missing-category recommendations and fallback ranking signals.
FR-06	Support optional customer favourites	Use the mobile number, when available and valid, to identify frequently bought products.
FR-07	Filter unavailable items	Exclude out-of-stock, off-menu, zero-price and already-in-cart products.
FR-08	Rank and diversify recommendations	Fuse normalised signals into a score and avoid a list dominated by one reason type.
FR-09	Explain each recommendation	Return a readable reason such as “Often ordered with ...” or “You ordered this before”.
FR-10	Provide staff insight output	Expose model statistics, buy-together rules and category attach rates in the staff application.
FR-11	Support public QR checkout recommendation requests	Allow anonymous requests using validated shop, branch, cart and optional phone context.

The QR customer workflow is functionally implemented and has been validated in the development environment. The recommendation dialog, the recommendation endpoint, the cart integration and the optional phone-based personalisation all exist in the implemented system. The QR ordering interface has not been deployed publicly, though. The design reflects this by treating the feature as an implemented client of the recommendation service and not as a service already running in production.

4.1.2 Non-functional and quality requirements

The component also has quality requirements that shaped the architecture. The system is meant for micro and small businesses, so the design favours techniques that are cheap to compute, services that are modular, ranking that is deterministic, explanations that are clear, and reuse of existing POS data in place of complex customer accounts or costly infrastructure. Table 4.2 sets out the nine non-functional requirements (NFR).

Table 4.2: Non-functional and quality requirements

ID	Quality requirement	Design response
NFR-01	Relevance	Recommendations should reflect the current basket and branch-specific evidence, not only global popularity.
NFR-02	Explainability	Each recommendation should include a reason that customers and staff can understand.
NFR-03	Availability awareness	The system should not knowingly recommend products that are currently unavailable.
NFR-04	Determinism	For the same prepared input the ranking should be stable; product ID is the tie-breaker.
NFR-05	Graceful data thresholding	If fewer than 10 baskets exist, the model should report that it is not ready instead of showing weak patterns.
NFR-06	Branch isolation	Historical evidence, stock status and recommendations should stay scoped to the relevant shop and branch.
NFR-07	Maintainability	Recommendation mining should remain separate from core transaction-processing responsibilities.
NFR-08	Transparency for debugging	Returned items should expose support, confidence, lift, similarity, popularity and attach rate where available.
NFR-09	Usability	Customer suggestions should be addable straight from the dialog without a separate navigation flow.

4.2 Component Architecture within the Complete Solution

The recommender works inside the wider SmartCost architecture and is not a separate application. Two clients use the same analytical model: the Next.js QR ordering interface for customers and the Expo/React Native application for staff. Both connect through the Node.js/Express backend, which controls shop and branch context, validates input, retrieves operational data and calls the Python analysis service. Figure 4.1 shows the arrangement.

Smart Order Recommendation System within SmartCost POS

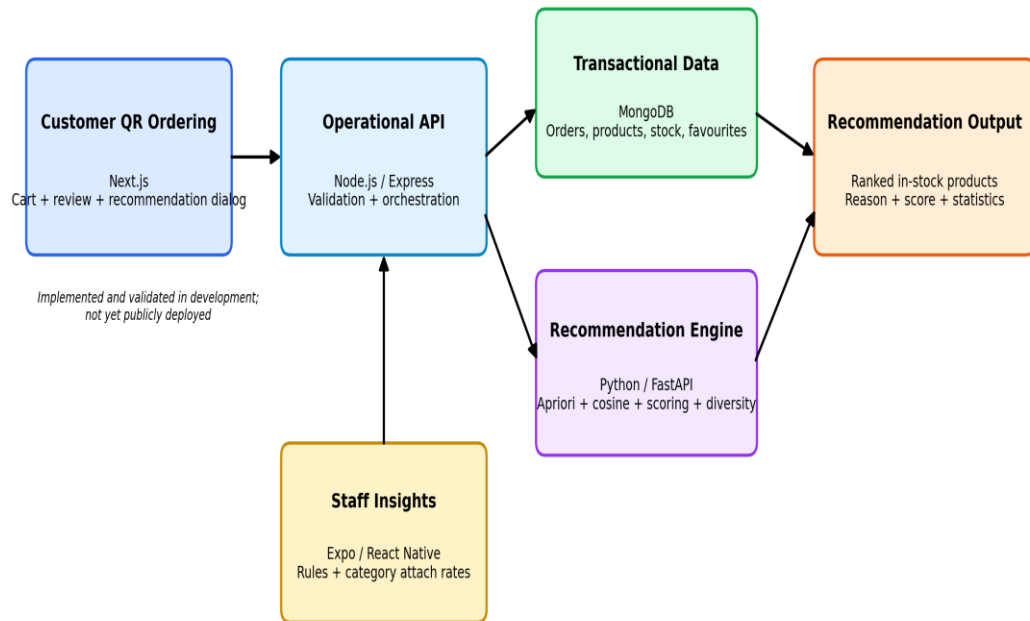


Figure 4.1: Smart Order Recommendation System within the complete SmartCost POS solution

The customer route is public, since QR customers do not authenticate before ordering. It validates the shop and branch, checks the submitted cart, accepts an optional phone number and passes the prepared request through the operational backend. The staff insights endpoint is protected by JWT authentication and takes the shop and branch from the authenticated user context.

MongoDB is the evidence store. The data-preparation service fetches three groups of information in parallel: completed order baskets, the currently orderable catalogue with branch stock, and optional customer favourites from purchase history. In the documented design the analysis service never queries the database directly. It receives a prepared request from Node.js. Data access and tenant context therefore stay in the main application, and the Python service stays focused on analysis. Table 4.3 lists the components and their responsibilities.

Table 4.3: Major components and responsibilities

Component	Technology	Primary responsibility
Customer QR interface	Next.js / Material UI	Builds the cart, opens the “Add something else?” dialog, sends the cart and optional phone number, and allows add, remove and skip actions.
Staff mobile interface	Expo / React Native / React Native Paper	Displays model statistics, buy-together patterns and category attach rates.
Operational API	Node.js / Express	Validates requests, prepares candidates, adds product metadata and reason sentences, and controls shop and branch context.
Data preparation service	Mongoose / MongoDB aggregation	Builds order baskets, retrieves the available catalogue, checks branch stock and derives favourites.
Analysis-service bridge	Axios / analysis service client	Forwards prepared recommendation payloads to the Python service.
Recommendation engine	Python / FastAPI	Performs Apriori mining, association rules, cosine similarity, attach-rate calculation, score fusion and diversity handling.
Database	MongoDB	Stores order history, products, branch stock and the data needed to derive customer favourites.

4.3 Design Alternatives and Decisions

The implemented design reflects several explicit trade-offs. I did not rely on one recommendation technique, because each has its own way of failing. Association rules are strong and explainable but need repeated co-occurrence. Item-item cosine similarity can expose weaker relationships that miss the rule-support threshold. Popularity and category attach rates give broader priors, and optional favourites add a little personalisation without a login account. Table 4.4 records the main alternatives I considered and what I chose.

Table 4.4: Design alternatives and implemented decisions

Decision area	Alternative considered	Implemented decision	Reasoning
Recommendation approach	Popularity-only ranking	Hybrid score fusion of association rules, cosine similarity, category attach rate, favourites and popularity	A popularity list ignores the cart; the hybrid combines contextual and fallback evidence.
Personalisation	Authenticated customer profiles	Optional mobile-number purchase history	QR customers stay anonymous while returning customers can still get limited personalisation.
Unavailable products	Recommend first, validate later	Filter candidates before ranking	Stops the system from showing products that cannot be ordered.
Analytical placement	Embed mining directly in Node.js	Separate Python FastAPI analysis service	Keeps operational API work apart from analysis and allows Python scientific libraries to be used.
Rule metric handling	Use raw lift in a linear score	Normalise lift as $\text{lift}/(1+\text{lift})$ before fusion	Stops unbounded lift from dominating bounded signals such as similarity and attach rate.
Multiple matching rules	Add every matching rule's contribution	Use only the strongest matching rule per candidate	Avoids a product collecting many weak rules and overtaking a genuinely strong recommendation.
Recommendation diversity	Pure top-score ranking	Replace the final slot when every top item has the same reason	Protects other recommendation types from being crowded out.
Small-data handling	Return whatever patterns exist	Return <code>modelReady=false</code> when fewer than 10 baskets exist	Avoids presenting unstable patterns as meaningful evidence.

Source: Author

The design also uses a 365-day lookback for order baskets. This is longer than a short forecasting window, because market-basket mining gains from transaction volume. Each order is a set of distinct product IDs and not a quantity-weighted vector, since Apriori asks whether products co-occurred in a basket, not how many units of each were sold.

A further decision was to use a branch-specific candidate pool. The catalogue is filtered to products with a positive price, and where inventory tracking is enabled the current branch stock is checked. Cart items are removed from the candidate set. The

ranking stage therefore only works with products that are plausible recommendations in the present operational context.

4.4 Detailed Design

The detailed design is a sequence of context collection, data preparation, pattern extraction, signal generation, score fusion, enrichment and output presentation. The Node.js backend prepares the operational context, and the Python service turns it into ranked recommendation evidence. Figure 4.2 shows the processing flow.

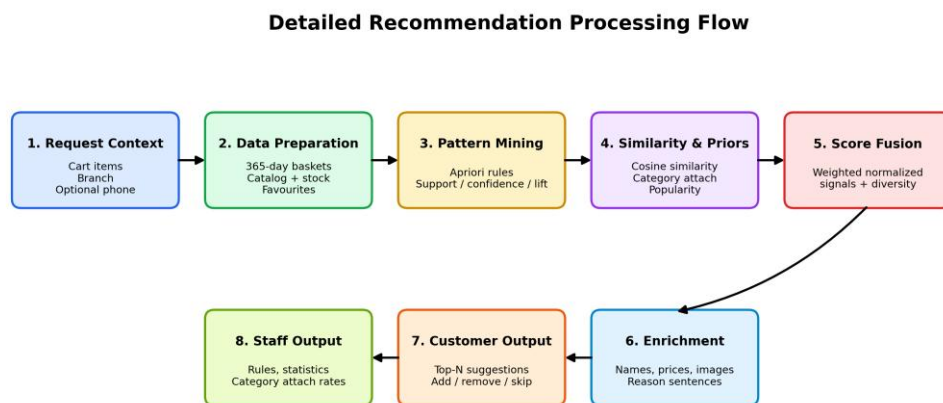


Figure 4.2: Detailed recommendation processing flow

Source: Author

4.4.1 Request and context design

A customer recommendation request contains the current cart lines, the shop identifier, the branch identifier, an optional mobile number and an optional result limit. The public endpoint validates malformed product IDs and clamps the limit to a controlled range. The backend then derives the categories of the cart items, so the analysis service knows which categories are already represented even though the cart products themselves are removed from the candidate list.

The payload sent to the analysis service holds the historical transactions as product-ID baskets, the cart item IDs, the cart category IDs, the candidate products with category metadata, optional favourite IDs and the model-tuning parameters. The documented

defaults are a minimum support count of 3, a maximum antecedent size of 2 and 8 nearest neighbours for item similarity. The internal API also accepts a maximum rule count, documented as 200.

4.4.2 Data preparation design

Historical evidence comes from completed History orders for the selected branch over the previous 365 days. Each order becomes a set of distinct product IDs, duplicates are removed and empty baskets are dropped. This suits frequent-itemset mining because the presence of an item in a basket matters more than its quantity for the Apriori logic used here.

The orderable catalogue is prepared separately. Products must be of type product and have a price above zero. Where branch-level inventory tracking applies, stock availability is checked before the product becomes a candidate, and candidates already in the cart are excluded. Filtering upstream reduces the risk that the analysis service ranks something that cannot be used.

Customer favourites are an optional data source. The only returning-customer signal is the mobile number entered by the QR customer. The backend derives the ten products most often ordered by that number at the same branch. If the number is missing or invalid, the request is still valid and personalisation is just left out.

4.4.3 Recommendation signal design

The engine combines five ranking signals. The first and strongest is a matching association rule. Apriori finds frequent itemsets and produces rules of the form $A \rightarrow B$, where the antecedent has one or two products and the consequent is the candidate product. Rules with lift of one or less are discarded because they show no positive association above chance. Table 4.5 lists the signals, their weights and when each is active.

Table 4.5: Recommendation signals used in score fusion

Signal	Normalised value	Weight	Activation condition
Frequently bought together	Best matching association-rule strength	1.00	All items in the antecedent are present in the cart.
Personal favourite	Binary favourite membership	0.85	The candidate appears among the returning customer's favourites.
Similar taste	Highest item-item cosine similarity to a cart item	0.70	The candidate is one of the top neighbours of a cart item.
Popular in category	Category attach rate	0.60	The candidate belongs to a category not yet represented and is a top product in that category.
Popular overall	Single-item support / popularity	0.35	The candidate has enough historical order frequency.

Source: Author

Since lift is unbounded and similarity and attach rates fall between zero and one, the engine compresses lift before fusion. Rule strength is $0.5 \times \text{lift} / (1 + \text{lift}) + 0.5 \times \text{confidence}$, and only the strongest matching rule contributes for a candidate. The result is a bounded rule contribution that can be combined sensibly with the other normalised signals.

A candidate's final score is the weighted sum of its active signal values. The reason shown to the user is the signal with the largest contribution. Candidates are sorted by descending score with product ID as a deterministic tie-breaker. If all top-ranked items have the same reason, the diversity rule may give the final slot to the best candidate with a different reason.

4.4.4 Output and interface design

The Python service decides which products to recommend and why. The Node.js backend then adds operational metadata to the returned product IDs: name, price, image, stock information and category. It also turns machine-oriented reason codes into short customer-facing sentences. This keeps the analytical reason stable while the application controls the wording. Table 4.6 shows the mapping.

Table 4.6: Explainable recommendation reason mapping

Reason code	Customer-facing explanation
frequently_bought_together	Often ordered with <cart item>
similar_taste	Goes well with <cart item>
popular_in_category	Most loved <category>
popular_overall	Popular right now
personal_favourite	You ordered this before

Source: Author

In the customer QR flow the request is triggered when the customer selects “Review & send”. The “Add something else?” dialog shows each suggestion with its image, price and reason, along with controls to add or remove the item. The customer can also skip the suggestions and go on to review and send the order. Placing the recommendation at this point puts it at a natural decision step just before confirmation, and it does not interrupt browsing.

The staff interface uses the same model with an empty cart to give branch-level insight and not customer-specific suggestions. The screen reports model statistics, up to twenty high-lift buy-together rules written as readable statements, and category attach rates with representative top products. If fewer than ten completed orders exist, the interface shows a “still learning” state instead of weak patterns.

4.4.5 API design

The solution has separate interfaces for customer use, staff insight and internal analysis. This boundary design keeps public customer access, authenticated staff access and service-to-service communication apart. Table 4.7 summarises the endpoint responsibilities documented for the implemented system.

Table 4.7: API boundaries of the recommendation component

Endpoint	Access	Design responsibility
POST /api/customer-orders/:shopId/:branchId/recommendations	Public	Validates shop, branch, cart and optional phone; returns enriched recommendations.
GET /api/recommendations/insights	JWT protected	Returns model statistics, buy-together rules and category attach rates for the authenticated shop and branch.
POST /recommendations	Internal FastAPI service	Runs mining, similarity, attach-rate calculation, scoring and ranking, and returns the analytical outputs.

Source: Author

4.5 Standards and Professional Practice

ISO/IEC 25010:2023 [30] supplies a product-quality vocabulary for assessing attributes such as reliability, security, maintainability and performance efficiency. The recommendation service’s future acceptance criteria should translate those attributes into testable targets, including response-time budgets, branch isolation, operational availability and recoverability. This is a proposed assessment framework, not a completed ISO certification.

The design follows software-engineering practices that are visible in the architecture: separation of concerns, explicit API boundaries, server-side validation, deterministic ranking, branch-level data isolation, stock-aware filtering, authenticated staff access and explainable output. These practices help maintainability and lower the risk of showing misleading or unusable recommendations at checkout.

Security responsibilities differ by interface. The public QR endpoint deliberately requires no authentication, since it serves anonymous table-order customers, so shop and branch identifiers are validated before any recommendation processing. The staff endpoint is protected by a JWT bearer token and takes its shop and branch from the authenticated session. The internal analysis endpoint is meant for service-to-service use through the application backend and not for direct customer interaction.

Privacy is kept low by not requiring customer accounts in the QR flow. Personalisation uses an optional mobile number and is skipped when it is absent or invalid. The documentation does not claim formal anonymisation, certified encryption or

compliance with a named privacy standard, so this thesis does not present such controls as implemented. It describes only the privacy-relevant behaviour that the implementation directly supports.

Responsible behaviour comes from explainable reasons, availability filtering, a minimum-data readiness threshold and the exposure of rule statistics to staff. These do not make the engine risk-free, but they reduce opacity and let business users inspect why particular product relationships were learned. The system also avoids showing low-data patterns when fewer than ten baskets exist.

Professional reproducibility is supported by the synthetic seed-data utility. The seed script can generate 12,000 deterministic synthetic orders with planted co-purchase patterns and a fixed random seed, which makes behaviour repeatable across development runs and gives a controlled basis for checking that the model rediscovers expected relationships. The same documentation records current limitations: per-request re-mining, a fixed minimum support, hand-tuned weights, no cached models, no stored recommendation uptake and no production fallback if the Python service is unavailable.

No claim is made that the implementation is certified against the International Organization for Standardization (ISO) standards, the Payment Card Industry Data Security Standard (PCI DSS) or any other named industry standard, because the project documentation gives no evidence of certification. “Standards and professional practice” here means the engineering conventions and safeguards that are demonstrably part of the design. A formal compliance assessment is outside the scope of this research.

5 IMPLEMENTATION, INTEGRATION AND TESTING

This chapter explains how the Smart Order Recommendation System was implemented as part of the SmartCost POS solution, how it was connected to the existing customer and staff workflows, and how its behaviour was checked. It concentrates on the real software artefacts and the actual execution path. The Node.js backend keeps responsibility for shop, branch, product, stock and customer-context handling, and the Python FastAPI service does basket mining, similarity analysis, score fusion and ranking.

Two interfaces use the implementation. The customer-facing one is the Next.js QR ordering page, where recommendations are requested just before an order is reviewed and sent. The staff-facing one is the React Native/Expo SmartCost application, where users can inspect model statistics, buy-together patterns and category attach rates. Both rely on the same recommendation engine, which avoids duplicated analytical logic and keeps behaviour consistent between the customer and staff views.

At the time of writing, the QR ordering module is functionally implemented and has been checked in the development and testing environment, but it has not been deployed to a public production host. I therefore keep functional implementation and production deployment separate in this chapter. The recommendation logic, the API communication, the interface flow and the branch-aware data handling can be demonstrated in the working development system. Public hosting and large-scale production operation are outside the completed deployment scope.

5.1 Development Environment and Technology Stack

The component was developed inside the existing SmartCost multi-component environment. I used a modular stack so that the transaction-oriented POS services and the analytical workload could change independently. The operational backend uses Node.js and Express, the database layer uses MongoDB through Mongoose, and the analysis service is written in Python with FastAPI. The customer ordering page uses Next.js and Material UI, and the staff application uses Expo/React Native with React Native Paper. The platform technologies are documented in [14]–[17]. The

components talk to each other through Representational State Transfer (REST) endpoints. Table 5.1 lists the layers.

Table 5.1: Development environment and technology stack

Layer / tool	Technology	Role in the implemented system
Customer ordering interface	Next.js + Material UI	Displays the QR ordering menu, cart, review flow and recommendation dialog.
Staff mobile interface	Expo / React Native + React Native Paper	Displays recommendation insights, model statistics, rules and category attach rates.
Operational backend	Node.js + Express	Validates requests, prepares data, applies business context and orchestrates analytical calls.
Database access	MongoDB + Mongoose	Stores products, historical orders, branch stock and customer-related purchase evidence.
Analysis service	Python + FastAPI + Pydantic	Hosts the recommendation API and validates analytical request and response structures.
Numerical / machine-learning utilities	NumPy, SciPy, scikit-learn	Support sparse basket matrices and cosine-similarity computation.
HTTP communication	Axios / REST	Connects the Node.js backend to the Python analysis service.

Source: Author

The Node.js layer handles request validation, shop and branch resolution, candidate preparation, communication with MongoDB and the call to the Python service. Business rules that depend on live POS state, such as stock availability and the active branch, therefore stay in the operational backend. The Python layer receives a normalised request with transaction baskets, current cart items, candidate products, categories already in the cart and optional customer favourites. It applies the hybrid logic and returns ordered recommendations and model statistics.

The analytical service is stateless with respect to the POS database. Python does not query MongoDB directly. Node.js collects the transactional and product data and sends the needed structures to FastAPI. This reduces coupling between the engine and the database schema, makes the recommender easier to test with synthetic request payloads, and keeps access to operational data under the control of the main backend. Execution is branch-specific. Baskets are retrieved for the active shop and branch, and candidate items are filtered with the same branch context. This matters because a product may be available at one branch and not at another, so the output reflects the state of the branch that made the request and not a global catalogue.

For learning, the completed order baskets from the preceding 365 days are used. Each order becomes a set of distinct product IDs, because the association and similarity logic is based on whether a product is present in a basket and not on quantity. A repeated item in one order therefore cannot inflate co-occurrence counts. There is also a minimum evidence rule: with fewer than ten completed baskets the analysis service reports that the model is not ready and does not produce recommendations.

5.2 Component Implementation

The component can be split into data preparation, recommendation computation, recommendation enrichment and interface presentation. Each of these is handled by a dedicated group of source files, so the model can be changed without rewriting the customer checkout controller, and the user interface (UI) can change without touching the analytical engine. Table 5.2 lists the main artefacts.

Table 5.2: Main implementation artefacts

Implementation area	Primary artefact	Responsibility
Staff API route	backend/src/novelty/novelty04RecommendationSystem/recommendationRoutes.js	Exposes the protected staff insights endpoint.
Customer API route	backend/src/routes/customerOrderRoutes.js and customerOrderController.js	Accepts anonymous QR recommendation requests and validates cart and phone input.
Data preparation	recommendationDataService.js	Builds baskets, stock-aware catalogue candidates and customer favourites.
Recommendation orchestration	recommendationController.js	Builds the analytical payload, calls Python, enriches recommendations and generates readable reasons.
Analysis-service bridge	recommendationClientViaML.js / analysisServiceClient.js	Sends recommendation payloads to the FastAPI service.

Implementation area	Primary artefact	Responsibility
Recommendation engine	analysis-backend/app/novelty04RecommendationSystem/recommender.py	Performs Apriori, association rules, item-item similarity, scoring and diversity.
FastAPI contract	router.py / schemas.py	Defines POST /recommendations and Pydantic validation.
Customer UI	dashboard/src/app/order/[shopId]/[branchId]/page.tsx	Displays the “Add something else?” recommendation interaction.
Staff UI	RecommendationsSection.tsx	Displays statistics, buy-together rules and category attach information.

Source: Author

5.2.1 Data preparation and candidate construction

For each request the backend prepares three forms of evidence: historical order baskets, an orderable product catalogue and optional customer favourites. Orders with completed status at the active branch are collected from the most recent 365 days. Each is turned into a basket of distinct product IDs, and empty baskets are removed before the payload goes to Python.

Candidate construction includes operational availability. Products must be valid saleable products with a positive price, and products that use inventory tracking are checked against branch stock. Items with zero available stock are excluded, and so are products already in the cart, since suggesting the same item again would add nothing to cross-selling.

The current cart is described by product IDs and by category IDs. The category information supports the category-gap signal. When the cart contains a main dish but no beverage or dessert, for example, the engine can favour strong candidates from a missing complementary category. This is close to what a waiter does when noticing what is missing from an order.

Optional personalisation comes from the mobile number entered in the QR flow. When a valid number is present, the backend finds the products most often ordered by that number at the current branch and sends up to ten favourite product IDs to the analysis service. Without a valid number, personalisation is skipped and the request does not fail.

5.2.2 Hybrid recommendation engine

The engine combines several complementary signals. Apriori mining finds recurrent product combinations and generates association rules from them. Rules have one- or two-item antecedents and a single-item consequent, because the consequent is the item that can be shown as a recommendation. Rules with lift of one or less are discarded because they do not describe a positive association beyond chance. The signals and their weights are those in Table 4.5.

The default minimum support count is three baskets. Frequent single items come first, then pairs and then triples, and downward-closure pruning avoids evaluating itemsets whose subsets are not frequent. The maximum antecedent size is two. From the surviving rules, at most 200 are kept after ordering by lift, confidence and support.

Item-item collaborative filtering complements the rules by finding weaker relationships that do not reach the minimum support needed for a rule. A binary basket-by-item sparse matrix is built and cosine similarity is computed between item columns, keeping the eight most similar neighbours of each item. This provides the “similar taste” signal, used when an item relates to the current basket although no strong association rule applies.

Three further contextual signals sit beside the learned rule and similarity evidence: category attach rate, overall popularity and personal favourites. Category attach rate is the share of historical orders that contain at least one item from a category. Popularity is single-item support. A personal favourite gives a direct prior when the optional customer history shows repeated purchases of the candidate.

Because lift is unbounded while cosine similarity, attach rate and popularity stay between zero and one, rule strength is normalised before fusion, as set out in Chapter

4. Each candidate then receives the weighted sum of all applicable normalised signals, and the explanation comes from the signal that contributes the most to its score.

A diversity rule is applied after ranking. If all selected recommendations share the same reason, the last slot goes to the best-scoring alternative with a different reason. A strong family of association rules therefore cannot completely crowd out a personal favourite, a category-gap candidate or a similarity-based suggestion. The default output size is ten, and the customer endpoint limits any requested size to between one and twelve items. Figure 5.1 shows the execution path from request to recommendation.

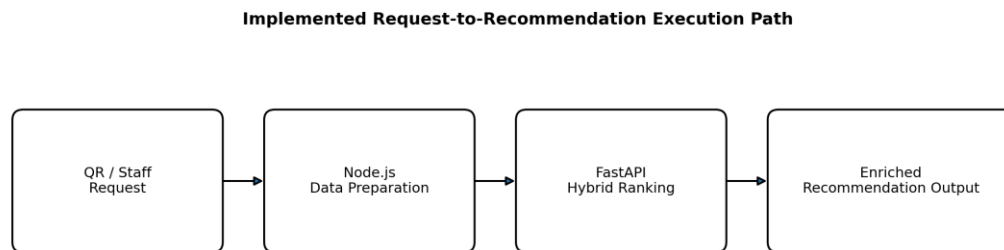


Figure 5.1: Implemented request-to-recommendation execution path
Source: Author

5.2.3 Output enrichment and explainability

The Python service returns product IDs, scores, reason codes and supporting statistics. It does not build customer-facing product cards. The Node.js controller enriches each selected product from the operational catalogue, adding the current name, price, image, category and availability. Ranking decisions therefore stay in the analytical engine, while the POS backend controls how the results look.

Explainability rests on a small set of stable reason codes. The code is chosen according to the strongest contribution to the candidate's score, and Node.js maps it to readable text. The customer sees why an item is suggested, and the developer can trace the displayed reason back to the signal that dominated the ranking.

Each recommendation can also expose its support, confidence, lift, cosine similarity, popularity and category attach rate alongside the final score. During development these values show whether a recommendation came from a strong co-purchase relationship, weaker item similarity, broad popularity, category context or customer history.

5.3 Integration with the Other Components

The recommender is integrated into SmartCost at API and user-interface level. It does not run as a standalone application. The integration starts with the shared MongoDB data model, continues through the Node.js business logic to the Python analytics service, and ends at the customer and staff interfaces.

5.3.1 Customer QR ordering integration

POST /api/customer-orders/:shopId:branchId/recommendations is the customer endpoint is. It is public on purpose, because QR customers are not asked to create or sign in to an account. The shop and branch identifiers in the route are validated, the cart lines are checked as product identifiers, and an optional phone number can be supplied to enable the personal-favourite signal. The Node.js backend then prepares the branch-specific analytical request and forwards it to the Python service.

The request is triggered at the checkout decision point and not while the customer browses. After building a cart, the customer selects “Review & send”. The interface opens the “Add something else?” dialog (Figure 5.2) and asks for recommendations for the current basket. Each returned suggestion can show an image, product name, price and explanation, and can be added to or removed from the cart before the customer continues.

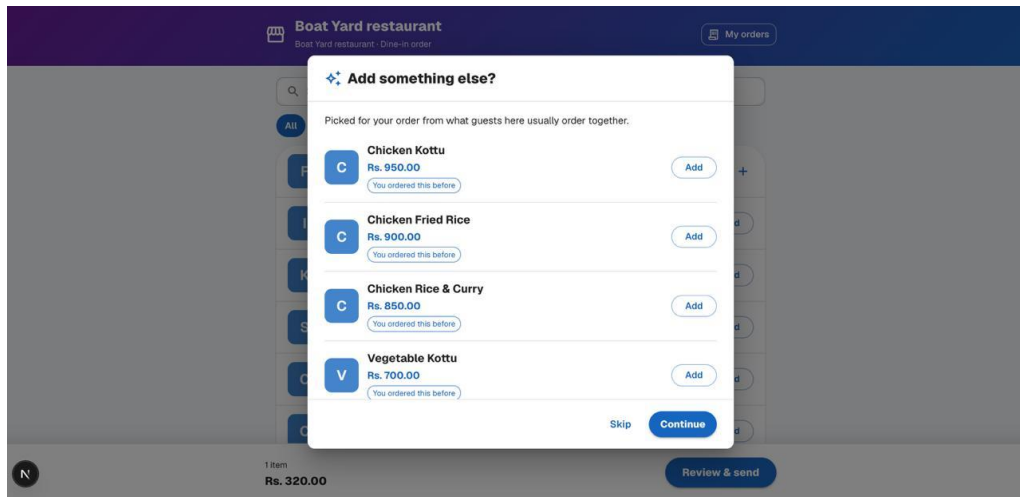


Figure 5.2: Customer recommendation dialog in the QR ordering workflow

After the recommendation step the customer continues to the usual order review and submission, shown in Figure 5.3. The existing order flow is preserved: the component proposes extra products but does not replace the transaction-processing logic of SmartCost.

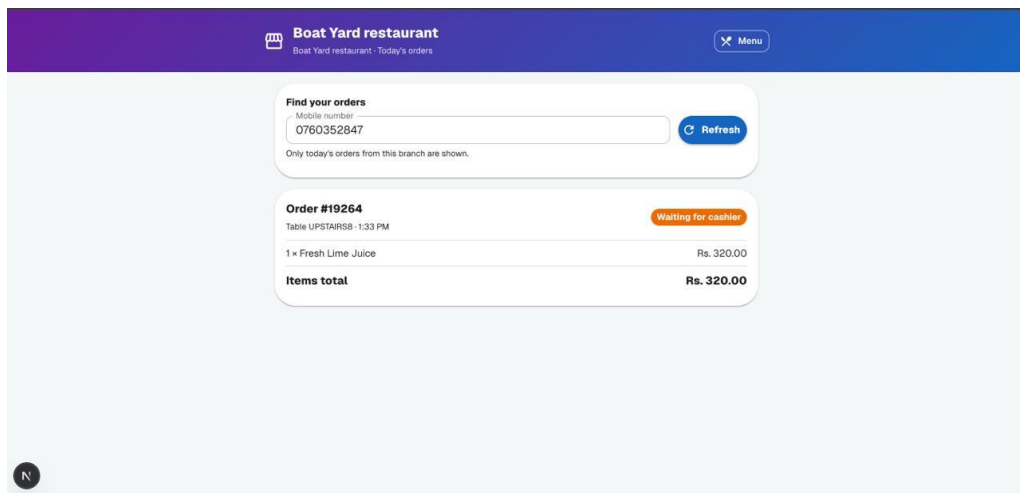


Figure 5.3: Customer order review stage after recommendation interaction

5.3.2 Staff mobile application integration

The staff integration uses the protected GET `/api/recommendations/insights` endpoint. Unlike the customer request, it takes the current shop and branch from the authenticated context. It runs the same analytical model with an empty cart, so the output describes purchase behaviour across the branch and not a single checkout.

The staff Recommendations screen shows model statistics such as the number of transactions learned from, the number of products represented, the number of rules kept, the average basket size and the model coverage. It also lists high-lift buy-together rules and category attach rates in readable language. Rules that refer to products which can no longer be ordered are removed before presentation so that the screen stays actionable. Figure 5.4 and Figure 5.5 show the two views.

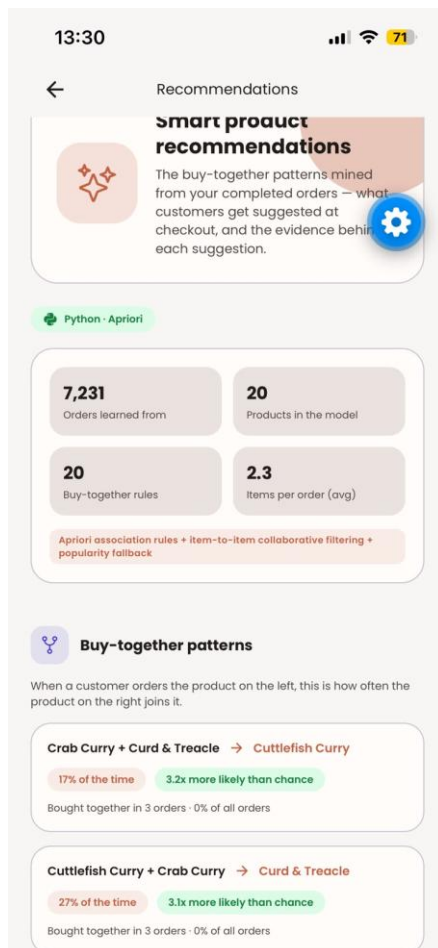


Figure 5.4: Staff recommendation model summary screen

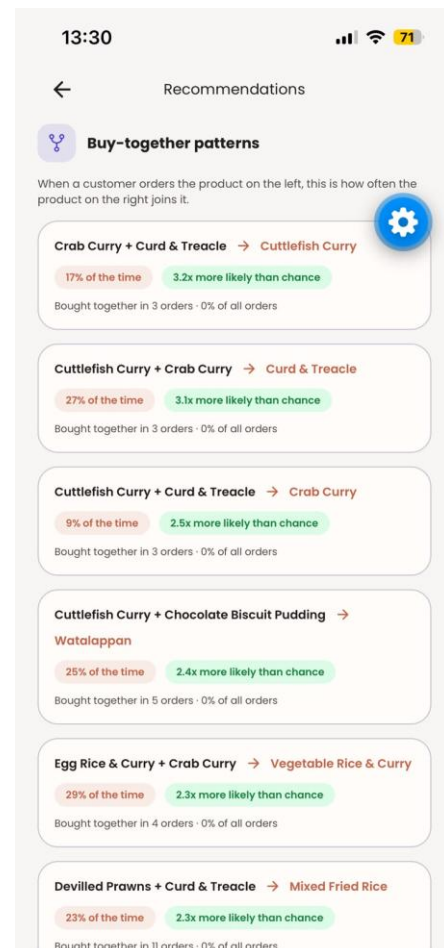


Figure 5.5: Staff view of learned recommendation patterns

5.3.3 Integration with manual orders and billing

The documented operating flow also shows how customer-originated orders connect to the existing manual-order and billing workflow. Staff open the manual-orders area of the SmartCost application (Figure 5.6) and use the QR and manual-order controls

to find incoming orders. Once the order is accepted and opened, the cart is prepared for normal POS processing, and the order continues to the kitchen and counter printing and billing path already provided by the wider system.

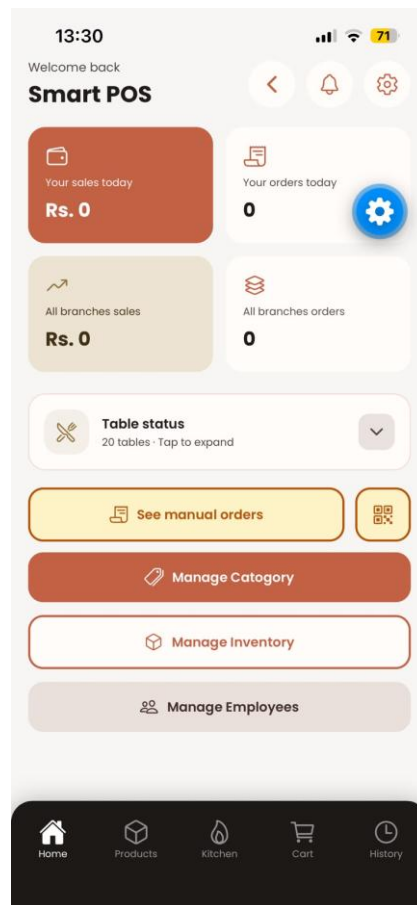


Figure 5.6: SmartCost mobile interface entry point for manual orders

The QR code screen in Figure 5.7 supports this workflow by giving customers a simple way into the restaurant ordering interface. Each QR code is tied to the relevant restaurant, shop and branch, so the customer reaches the matching menu without installing a separate app. After scanning the code with a phone, the customer goes to the web ordering interface, where products can be selected, reviewed and submitted.

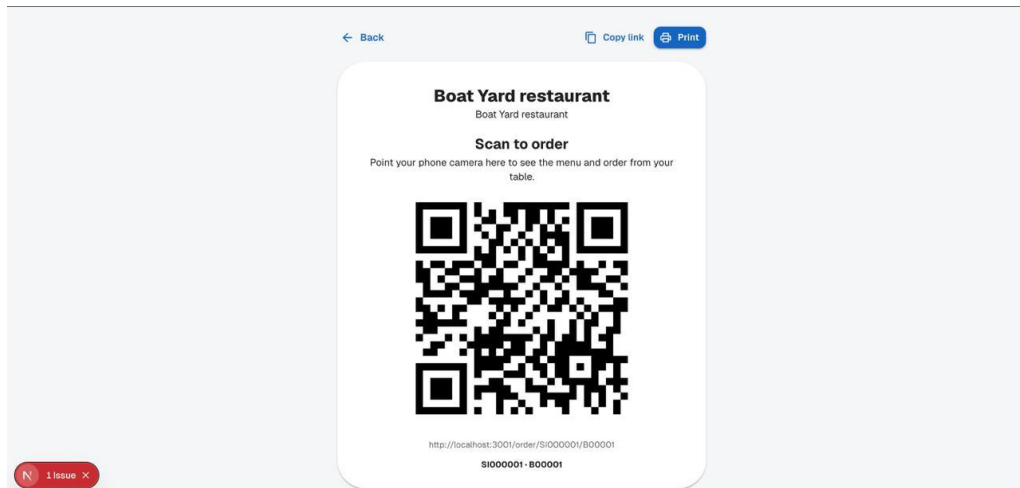


Figure 5.7: QR code screen used within the manual-order workflow

This QR-based flow is also the customer's entry point to recommendation generation. When the customer moves on to review the chosen items, the current cart is sent to the recommendation service so that suitable complementary products can be suggested before final submission. The recommendation step stays inside the ordering sequence and is not a separate analytical function.

The QR ordering flow is functionally implemented and has been validated in the development and testing environment. At the time of this study, the customer-facing QR interface had not been deployed to a public production host. The implementation demonstrates the complete interaction and integration flow under controlled development conditions, and production hosting remains part of the deployment work. Figure 5.8 and Figure 5.9 show the manual-order list and the confirmation after an order is accepted.

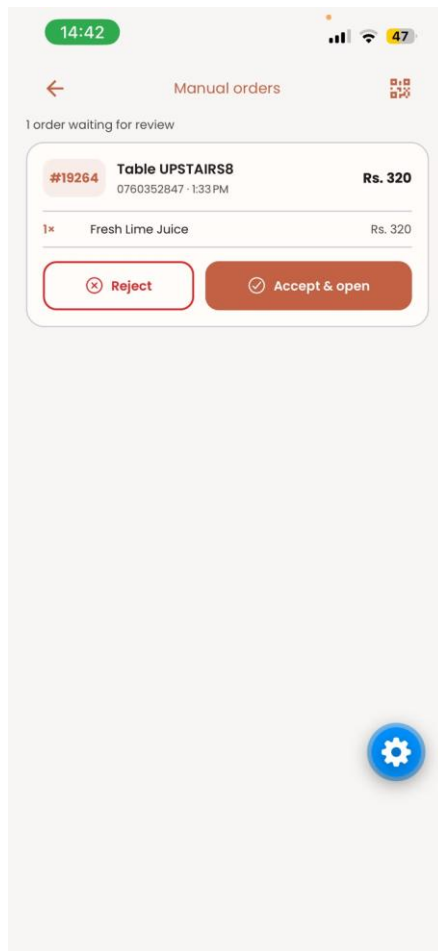


Figure 5.8: Manual-order list before accepting an order

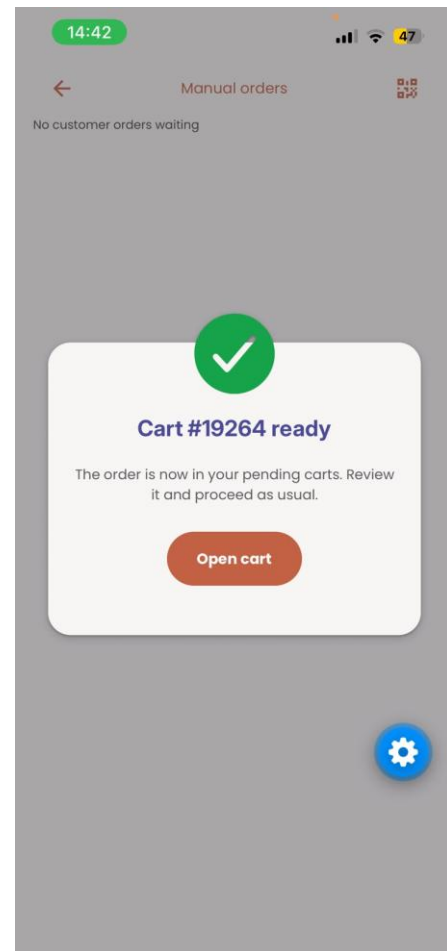


Figure 5.9: Confirmation after accepting and opening the order

The recommender therefore works with the billing component and does not duplicate it. Its responsibility ends once it has supplied the customer's chosen cart state to the order workflow. Kitchen routing, counter handling and bill printing belong to the wider SmartCost transaction process.

5.4 Verification and Software Testing

Verification was done at several levels: deterministic validation of the recommendation engine, checks of API behaviour, checks of the interface flow, and integration testing across the customer, backend, analytics and staff layers. The main test dataset is generated by the seedRecommendationData.js utility, run with the command `npm run seed:recommendations`. It produces synthetic orders with known

planted co-purchase relationships, so the engine can be checked against patterns whose expected direction is known beforehand. Table 5.3 gives the configuration.

Table 5.3: Deterministic test-data configuration

Test-data parameter	Implemented value
Generated order volume	12,000 orders over 180 days, configurable
Target context	Shop SI000001 and branch B00001 by default
Lunch / dinner distribution	55% lunch, 45% dinner
Typical basket opener	Kottu 45%, Rice & Curry 34%, Seafood 13%, drinks or dessert only 8%
Random noise	One random extra item in 30% of orders; two in 8%
Random seed	20260827, using a deterministic generator
Data safety	--clean deletes only rows tagged as recommendation-system seed data; --dry-run previews samples

The seed data contains several ground-truth patterns, listed in Table 5.4. Kottu or Fried Rice is attached to a beverage with 70% probability. Rice & Curry is attached to a seafood side with 45% probability. Any main dish is attached to a dessert with 35% probability. Grilled or devilled seafood is attached to Fresh Lime Juice with 60% probability. Putting much of each category-level pattern on a signature item makes sure Apriori can detect a meaningful item-level association and the probability is not spread too thinly across many combinations.

Table 5.4: Planted patterns used for behavioural validation

Ground-truth pattern	Attach probability	Verification expectation
Kottu / Fried Rice → beverage	70%	A beverage partner should emerge as a strong positive rule.
Rice & Curry → seafood side	45%	A seafood-side relationship should be rediscovered with lift > 1.
Any main → dessert	35%	Dessert category attach rate and product suggestions should increase.
Grilled / devilled seafood → Fresh Lime Juice	60%	Fresh Lime Juice should rank strongly for matching seafood baskets.

5.4.1 Algorithm and ranking verification

Algorithm-level verification checks that the implementation respects the configured thresholds and ranking rules. With the default configuration, an item or itemset must occur in at least three baskets to take part in frequent-itemset mining, rule antecedents are limited to two products, eight cosine neighbours are kept for each item, and at most 200 positive-lift rules are returned. When there are fewer than ten baskets, `modelReady` is false and recommendations are deliberately withheld.

Ranking verification checks that a candidate already in the cart is not recommended, that an unavailable tracked-stock item is excluded, that a matching personal favourite receives the personal-favourite signal, that a category absent from the cart can receive a category-attach contribution, and that the final explanation matches the signal with the greatest contribution. The diversity rule is checked by creating cases where many high-scoring candidates share one reason and confirming that the final list contains at least one alternative reason whenever an eligible candidate exists.

5.4.2 API verification

The customer API is checked with valid and invalid product IDs, with and without an optional phone number, and with different limits. The endpoint accepts anonymous requests after shop and branch validation, clamps the limit to the supported range of one to twelve, and rejects malformed product IDs with an `ITEM_INVALID` response. A failure of the Python analysis service passes through as a server error, since the current implementation has no fallback engine. Table 5.5 lists representative cases and the behaviour expected of each.

Table 5.5: Representative API verification cases

Test case	Expected behaviour
Valid branch, valid cart, sufficient history	Returns success, <code>modelReady=true</code> and ranked recommendations.
Valid cart but fewer than ten historical baskets	Returns <code>modelReady=false</code> and no recommendations.
Malformed product identifier	Returns 400 <code>ITEM_INVALID</code> .
Phone omitted or invalid	Recommendation still succeeds without the personal-favourite signal.

Test case	Expected behaviour
Tracked item has zero branch stock	Item is excluded from candidates and cannot be recommended.
Requested limit below or above the supported range	Limit is clamped to 1–12.
Python service unavailable	Current implementation returns a server error; no fallback engine is implemented.
Staff endpoint without the required authenticated branch context	Request is rejected according to protected-endpoint validation.

5.4.3 Interface and end-to-end integration testing

End-to-end testing follows the sequence a real customer and staff member would use. A customer opens the restaurant menu, selects products, goes to review and receives the recommendation dialog. More items can be added before the order is submitted. On the staff side, the matching order is found through the manual-order workflow, accepted and opened, after which normal cart, kitchen and billing processing continues.

The screenshots in this chapter are the implementation evidence for these stages. They show that recommendation generation is connected to the customer ordering interface and that downstream staff handling can receive and continue processing the resulting order. This validation is functional and is not a production-load test, because the QR ordering module has not been publicly hosted.

5.5 Deployment and Operation

The recommendation component works in the development and testing environment. The runtime design assumes that the Node.js backend can reach the Python FastAPI analysis service and that both can reach the application data needed to prepare requests. The customer QR page and the staff application talk to the backend through REST endpoints, and the backend talks to FastAPI through an internal HTTP request.

Production deployment of the QR ordering interface has not been completed, so this thesis makes no claim of public availability or production-scale traffic results. The behaviour has been demonstrated in the working development environment, including

the customer recommendation dialog, the order review flow, the staff recommendation-insights screen and the manual-order handling path.

Several operational limitations affect deployment readiness. The model is currently mined again on every request from the available historical baskets, with no cached or precomputed rules and similarity matrices. This keeps the implementation simple and the results current, but response time can grow with transaction volume. The Python service is also a single analytical dependency, and when it is unavailable the backend does not fall back to a simpler popularity-only answer.

The system also does not store acceptedRecommendations from the QR page in the backend transaction record. The client can send the accepted recommendation information, but because it is not kept, the system cannot yet compute real recommendation uptake, incremental revenue or changes in average order value. I therefore treat these measures as future operational analytics and not as completed production metrics.

Operational improvement should include branch-level caching of mined rules and the similarity matrix, scheduled refresh of recommendation artefacts, a fallback response when the analysis service is unavailable, monitoring of response time and service health, and persistent storage of accepted recommendation events. Public hosting would also need Hypertext Transfer Protocol Secure (HTTPS). These would move the working recommendation flow towards a production deployment that can be relied on while keeping the same API contracts and user experience. Table 5.6 compares the current state with the production-oriented improvements.

Table 5.6: Current operational state and deployment improvements

Operational aspect	Current state	Production-oriented improvement
QR ordering functionality	Implemented and validated in the development and test environment	Deploy to public production hosting with HTTPS and monitoring.
Recommendation computation	Mined on each request	Cache rules and similarity per branch and refresh on a schedule.
Analysis-service failure	No fallback engine	Provide a graceful popularity or category fallback.

Operational aspect	Current state	Production-oriented improvement
Recommendation uptake tracking	acceptedRecommendations not stored	Store acceptance events and calculate uptake and revenue metrics.
Model thresholds	Fixed defaults	Adapt the support threshold to branch transaction volume.
Context features	No time, day or order-type context	Add contextual signals once enough operational data is available.

Overall, the implementation shows complete functional integration of branch-aware data preparation, hybrid recommendation computation, explainable ranking and customer and staff presentation in the SmartCost development environment. What remains is mostly production hardening, monitoring and measurement, and not the basic recommendation function.

6 RESULTS AND VALIDATION

This chapter reports the results that the current implementation and its technical artefacts can support. The evidence comes from the hybrid recommendation pipeline, the deterministic synthetic validation data, observable API behaviour, the customer and staff interfaces, and integration with the wider SmartCost POS workflow. The group’s earlier research paper described an association-rule-focused evaluation and gave numerical outcomes that the current artefacts do not reproduce, so I discuss that paper separately wherever it is relevant.

The implementation combines association-rule mining, item-item cosine similarity, category attach rates, popularity, optional customer favourites, stock-aware candidate filtering and a diversity rule. Validation is therefore wider than the test of one algorithm. The main question is whether the component can turn branch-specific transactional evidence and live POS context into useful, explainable and operationally valid recommendations, and whether it exposes enough statistics for staff and developers to inspect.

A second point is the difference between functional validation and measuring production impact. The flow works in the development and testing environment, and the synthetic seeding process gives reproducible ground-truth patterns for checking the analytical pipeline. Accepted recommendations are not stored by the backend, however, and no hold-out Precision@K or Hit Rate evaluation exists in the codebase. I do not present production uplift, real-world acceptance rate or offline recommendation accuracy as findings of the current implementation.

6.1 Research Results

The main research result is a working hybrid market-basket recommendation mechanism that uses several complementary kinds of evidence and not a single popularity list. Historical completed orders are converted into sets of distinct product IDs, and the analytical service searches them for frequent itemsets and association rules. The same baskets also produce an item-item similarity structure, while category

participation and product frequency supply broader priors. These signals are combined with optional customer favourites and the current cart to give a final ranked list.

The association-rule component produces interpretable relationships of the form $A \rightarrow B$. Rule quality is expressed through support, confidence and lift, and rules with lift of one or less are discarded because they show no positive relationship beyond chance. The implementation keeps at most 200 rules and uses antecedents of up to two products. This limits the search space and still allows combinations such as a main item plus a side leading to a further recommended product.

The second result is the use of item-item cosine similarity as a complement. Association rules can miss weaker but meaningful co-purchase relationships when a product pair does not reach the minimum support threshold. Using a binary basket-by-item representation, the system measures how similarly two products take part in historical orders. Up to eight nearest neighbours are kept for each item, which lets the recommender produce a “Goes well with” explanation even when no strong association rule exists.

The third result concerns contextual score fusion. The system does not add raw measurements with incompatible scales. It normalises each signal’s contribution and then applies a fixed design weight. Rule strength comes from a bounded transformation of lift combined with confidence, whereas cosine similarity, category attach rate, favourites and popularity are already bounded or binary. The final score is the weighted sum of the applicable contributions, and the explanation shown to the user comes from the strongest contributor, which ties the analytical decision to a readable reason. Table 6.1 summarises the research outputs.

Table 6.1: Research outputs supported by the current implementation

Research aspect	Implemented result	Validation evidence
Association discovery	Apriori frequent itemsets and $A \rightarrow B$ rules	Support, confidence, lift and rule count are exposed.
Similarity modelling	Item-item cosine neighbours	Top neighbours can contribute to candidate scores.
Context awareness	Current cart, represented categories, branch and optional favourites	Request payload and branch-aware data preparation.
Operational validity	Out-of-stock and already-selected items excluded	Candidate construction uses branch stock and cart filtering.

Research aspect	Implemented result	Validation evidence
Explainability	Reason code and readable reason for each recommendation	Output exposes score, reason and underlying statistics.
Diversity	A different reason can replace the final slot when results are uniform	Explicit diversity post-processing rule.

The deterministic seed dataset is the strongest direct validation of the research logic at present. The seed script generates 12,000 synthetic orders over a configurable 180-day period using a fixed random seed. It deliberately embeds known co-purchase structures: a strong beverage attachment to Kottu or Fried Rice, a seafood-side relationship for Rice and Curry, dessert attachment to main dishes, and a Fresh Lime Juice relationship for grilled or devilled seafood (Table 5.4). Since these structures are planted, the system can be checked to see whether the mining process rediscovers them with positive rule strength. Figure 6.1 shows the validation strategy.

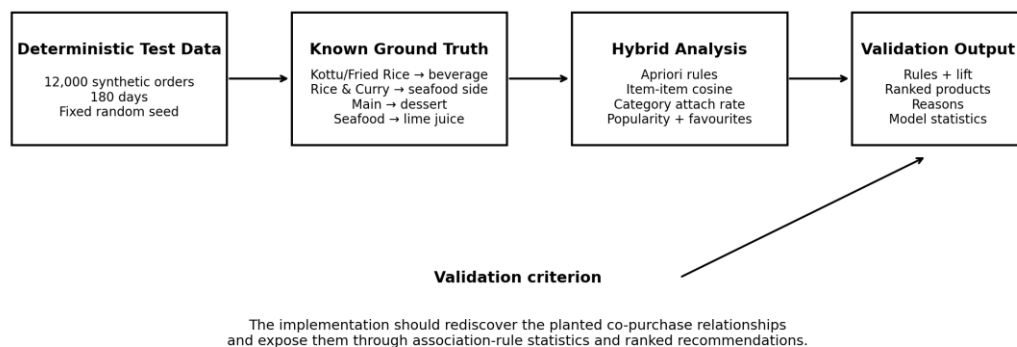


Figure 6.1: Ground-truth validation strategy for the Smart Order Recommendation System

This approach shows whether the mining and ranking pipeline behaves consistently with a known input structure. It should not be read as a replacement for an offline recommender benchmark on unseen real orders. The technical documentation names hold-out measures such as Precision@K and Hit Rate as future work. At this stage the research result is evidence of correct pattern recovery and functional ranking behaviour, and not a statistically complete measure of how well the model generalises.

6.2 Product and Technical Results

At product level, the result is a recommendation capability built into two SmartCost interfaces. In the customer workflow, the Next.js QR ordering page requests recommendations just before the user completes order review. The returned items can include the product image, price, category, availability, a readable reason and analytical statistics. The customer can add or remove recommended items in the dialog before continuing with the order. The analytical output has thus become an action at checkout and is not confined to an administrative report.

The second product result is the staff recommendation-insight view in the mobile POS application. The staff endpoint runs the same model without a customer cart to summarise behaviour across the branch. The interface presents model statistics, buy-together rules and category attach rates. Rules are written as readable statements, and products no longer on the menu are left out of the insights, so owners and staff can see the patterns that influence what customers are offered. The staff screen captured in Figure 5.4 and Figure 5.5 gives a concrete example from the development branch. It reports that the model learned from 7,231 orders covering 20 products, kept 20 buy-together rules, and found an average of 2.3 items per order. The listed rules include, for instance, Crab Curry with Curd & Treacle leading to Cuttlefish Curry, shown as 3.2 times more likely than chance, and Egg Rice & Curry with Crab Curry leading to Vegetable Rice & Curry, shown at 2.3 times. Each rule also states how often the second product joins the first and in how many orders the combination was seen. These figures come from the development data on that screen. They show the pipeline producing readable, quantified rules, but they are not a measure of recommendation accuracy.

Technically, recommendations are returned through a structured API contract. The customer endpoint is public because QR users do not authenticate, but shop and branch identifiers are validated. The staff endpoint is protected by a bearer token and finds the active shop and branch from the authenticated context. The internal FastAPI endpoint returns the recommendation method, model readiness, minimum data requirement, model statistics, ranked recommendations, learned rules and category attach information. Because of this separation the operational backend stays responsible for

business data while the Python service does the analysis. Table 6.2 lists the observable outputs.

Table 6.2: Observable technical outputs of the recommendation component

Output	Meaning	Use
modelReady	Whether enough transaction evidence exists	Prevents recommendations from very small histories.
transactionCount	Number of baskets used by the model	Shows the amount of evidence available.
itemCount	Items meeting the minimum support	Describes the effective model catalogue.
ruleCount	Retained positive-lift association rules	Measures the volume of learned rules.
pairCount	Frequent product pairs identified	Describes the co-purchase structure.
avgBasketSize	Average distinct products per order	Provides basket context.
coverage	Share of recommendable items reachable through rules or neighbours	Indicates the breadth of recommendation reach.
score + reason	Rank and dominant explanatory signal	Supports customer explanation and debugging.

The system behaves predictably when evidence is thin. If fewer than ten completed baskets exist for the branch, the analytical service returns a not-ready state and does not show unstable recommendations. I consider this a significant technical result, since the product does not imply confidence where the historical evidence is too limited. A related constraint is that products ordered fewer times than the minimum support count may stay in a cold-start state and receive no support from the learned signals.

Stock-aware generation is another technical result. Candidate products are not chosen from the historical model alone. The current branch catalogue and BranchStock state are considered before candidates are sent for ranking, and items that are unavailable or already in the cart are excluded. The final output is therefore valid for the current transaction and not only statistically associated with past baskets.

6.3 User or Stakeholder Validation

Direct stakeholder validation in the current artefacts is more limited than the technical validation. The documentation demonstrates the customer recommendation dialog, the staff insight interface and the wider order-processing workflow. It does not contain a

raw questionnaire dataset or a participant-level usability analysis that could be reproduced. I therefore treat interface walkthroughs and stakeholder-facing functionality as demonstrable evidence, and I do not claim that the current artefacts independently reproduce a formal user-study result.

The group’s earlier research paper describes a practical evaluation with eight bakery and café cashiers and managers. According to that paper, participants used the recommendation-enabled checkout interface in simulated transactions and gave positive feedback on usefulness, clarity and fit with the workflow. They also valued explainable messages because these made the suggestions easier to follow. These observations are relevant to the design motivation of the current system, especially its use of explicit reason text.

The paper and the current documentation do represent different stages of the project, however. The paper describes an association-rule-focused evaluation and reports numerical recommendation and business-impact outcomes. The current component is explicitly hybrid, and its documentation states that accepted recommendations are not stored and that no hold-out accuracy metric is calculated. The earlier user-evaluation account is therefore presented as supporting historical evidence and not as independently verified quantitative validation of the present codebase. Table 6.3 sets out the evidence sources and how I interpret them.

Table 6.3: Stakeholder validation evidence and interpretation

Evidence source	What it supports	Interpretation in this thesis
Customer QR workflow screenshots	The recommendation interaction is understandable and built into checkout.	Functional and interface validation.
Staff recommendation screen	Business users can inspect rules, attach rates and model statistics.	Transparency and decision-support validation.
Supporting paper: 8 cashiers and managers	Positive usability and explainability observations were reported.	Supporting historical stakeholder evidence.
Current implementation documentation	No stored recommendation acceptance outcome.	Production uptake is not claimed as a validated result.

The stakeholder-oriented outcome that can be shown directly is explainability. Each recommendation can come with a reason such as “Often ordered with <cart item>”, “Goes well with <cart item>”, “Most loved <category>”, “Popular right now” or “You

ordered this before”. These statements map to the dominant analytical signal used in ranking, so the customer and the staff member get a meaningful justification and not an unexplained number.

6.4 Integration Results

Integration testing shows that the component works across the SmartCost application layers without the Python service accessing the POS database directly. The Node.js backend retrieves historical orders, current product details, categories, stock availability and optional customer favourites from MongoDB. It turns this operational state into a structured request for FastAPI, receives the ranked result, adds product-facing information and returns it to the right customer or staff interface.

The customer path starts from the QR ordering page when the user reaches the review stage. The cart product IDs and represented categories are sent to the recommendation endpoint, together with an optional stored mobile number for favourite-based personalisation. The engine returns ranked candidates, which the web interface shows in the “Add something else?” interaction. The user may add a recommendation and continue through the normal order-submission workflow, which shows that the component is part of checkout and not separate from it.

The staff path is authenticated and branch-aware. The mobile application calls the insights endpoint, and the backend finds the shop and branch from the protected request context. The same engine then produces branch-level model statistics, association rules and category attach information. One recommendation model is thus used for two audiences: customer upselling and staff decision support. Figure 6.2 shows the integrated results across the application layers, and Table 6.4 summarises each integration point.

Layer	Integrated input / responsibility	Observed integrated result
Customer QR interface	Cart context and optional phone	Recommendation cards with add/remove
Node.js / Express	Validation, branch context, enrichment	Public customer + protected staff APIs
MongoDB	Orders, products, categories, branch stock	365-day baskets and live availability
Python / FastAPI	Mining, similarity, score fusion	Ranked recommendations + model stats
Staff mobile app	Authenticated branch context	Rules, attach rates and model statistics

Figure 6.2: Integrated results across SmartCost application layers

Table 6.4: Integration validation summary

Integration point	Observed result	Status
Next.js → Node.js customer endpoint	Cart and optional phone can request recommendations.	Functionally implemented
Node.js → MongoDB	Branch-specific baskets, catalogue, stock and favourites are prepared.	Functionally implemented
Node.js → FastAPI	Recommendation payload is sent and the ranked result returned.	Functionally implemented
FastAPI → Node.js enrichment	Scores and reasons are converted into a product-facing response.	Functionally implemented
React Native/Expo → staff endpoint	Rules, model statistics and attach rates are displayed.	Functionally implemented
QR public hosting	Customer flow works in the development and testing environment.	Not yet publicly deployed

The integration work also exposes some operational limitations. The model is recomputed from historical baskets on each request and is not served from a cached representation. If the Python service is unavailable there is no fallback recommender, and the request can fail. These limitations do not undo the functional integration, but they mark the production-readiness work that remains for deployment and scaling.

The QR ordering interface is functionally implemented and checked in the development and testing environment, but it is not hosted as a public production service. The result shown in this thesis is therefore end-to-end functional integration in the development environment. It is not evidence of sustained public deployment under real restaurant traffic.

6.5 Summary of Findings

The results show that the Smart Order Recommendation System has moved beyond a conceptual association-rule prototype and is now an implemented hybrid recommendation component integrated with the SmartCost POS architecture. It can learn co-purchase evidence for a branch, add item-item similarity to strong association rules, account for category gaps and popularity, include optional returning-customer favourites, exclude unavailable products, and produce a ranked list with an explicit reason for each item.

Deterministic synthetic validation shows that the pipeline can be tested against known purchasing structures. The seed process generates 12,000 reproducible orders containing planted co-purchase relationships, which gives a controlled basis for judging whether the model rediscovers the expected patterns. Model statistics and rule measures add transparency about what the analytical service has learned.

At product level the recommendations are built into the customer QR checkout and a staff insight screen. The same analytical evidence therefore supports immediate cross-selling and longer-term staff understanding. The implementation is branch-aware and stock-aware, the operational backend remains the authority for POS data, and the Python service handles mining and ranking.

The evidence also sets the current boundaries of the research. The codebase does not store accepted recommendations, does not calculate a hold-out Precision@K or Hit Rate, and is not yet served through a public QR-ordering host. Production acceptance rate, incremental revenue and offline accuracy of the current model therefore cannot be claimed as validated outcomes. The earlier paper contains numerical accuracy, latency and user-study claims, but I treat them as supporting evidence from an earlier research stage and do not carry them over to the present hybrid implementation.

Overall, the findings support the central contribution: a lightweight, explainable and operationally integrated hybrid recommendation mechanism suited to a micro-enterprise POS context. The implementation shows that routine transaction history can be turned into contextual product suggestions while respecting practical constraints such as branch availability, transparent reasoning and limited customer identity requirements. The remaining gaps concern production-scale evaluation, caching, fallback operation and measurement of real recommendation uptake, which the discussion and future-work chapters take up.

7 DISCUSSION

This chapter interprets the findings of the Smart Order Recommendation System against the research problem, the recommendation approaches reviewed earlier, the SmartCost product context and the limitations found during validation. I separate what the current hybrid implementation demonstrates from what the group's earlier research paper reported. The code now combines association rules, item-item similarity, category attach rates, popularity and optional customer favourites, whereas the earlier paper described a stage of the work that centred on association rules.

The central result is more than a list of products. Recommendation generation can be built into a micro-enterprise POS workflow and remain branch-aware, stock-aware, explainable and usable by customers and staff alike. Historical order baskets become market-basket evidence, the operational catalogue and branch stock constrain the candidate set, and the current cart influences the final ranking. Analytical relevance and operational feasibility are joined in this way. Figure 7.1 presents the interpretive synthesis.

The evidence has to be read within the limits of a development-stage research prototype. Deterministic synthetic data verifies that known co-purchase structures can be rediscovered, and the integrated interfaces show that ranked recommendations reach both the checkout and the staff insight flows. The implementation does not yet store accepted recommendation outcomes, compute a hold-out Precision@K or Hit Rate, or run through a publicly deployed QR ordering service. The discussion therefore deals with feasibility, design contribution and verifiable analytical behaviour, and it makes no unsupported claim about production sales uplift.

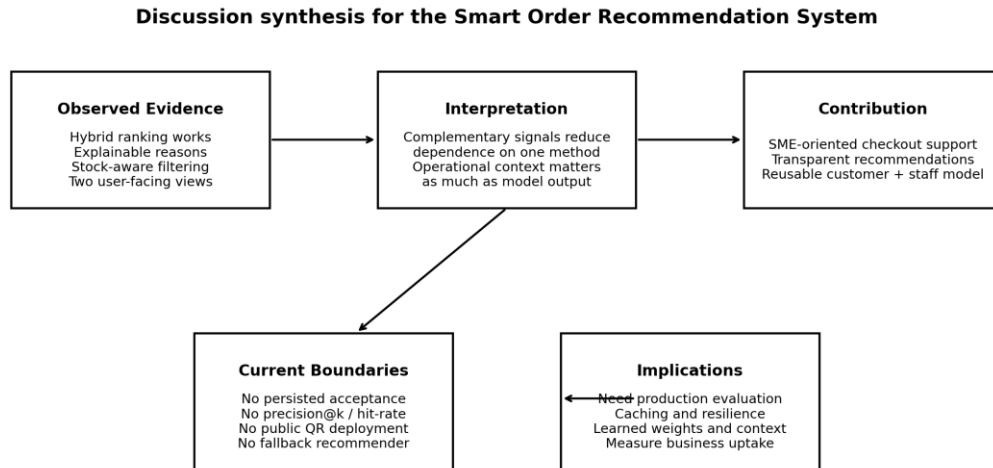


Figure 7.1: Interpretive synthesis of findings, contributions and current boundaries

7.1 Interpretation of Findings

The findings suggest that hybridisation is the most significant technical feature of the approach. Association rules identify strong and interpretable co-purchase patterns, but they depend on combinations reaching a minimum support threshold. Item-item cosine similarity widens the evidence by finding products with similar basket patterns even when a formal rule is weak or absent. Category attach rates and popularity add fallback-style priors, and optional customer favourites add a limited personal signal without a user account.

This layered design matters in a micro-enterprise because data density varies a lot between products and branches. A single method can fail for different reasons. Association rules are sparse when product pairs are infrequent, collaborative techniques are weak when customer histories are short, and plain popularity becomes generic and blind to the current basket. Combining several signals lets the system recommend from the best evidence available while keeping a clear link to the current order.

Branch-specific stock information changes what recommendation quality means, from a purely analytical idea to an operational one. A statistically strong suggestion is

useless if the item cannot be sold. Candidate preparation therefore removes unavailable products before ranking, so the validity of a recommendation depends on historical association strength and also on whether the item can be ordered at that branch when the request is made.

Explainability is another key finding. The system exposes a reason tied to the dominant contribution to each recommendation: an item frequently ordered with something in the cart, similar to a cart item, popular in a missing category, generally popular, or previously ordered by the same phone number. These messages connect analytical scoring and human interpretation. For owners, cashiers and QR customers this is more practical than an unexplained probability or ranking score.

The staff insight interface extends the value of the same model beyond immediate upselling. Called without a customer cart, the analysis describes patterns across the branch through model statistics, buy-together rules and category attach rates. The component is therefore both a transactional recommender and a light business-intelligence tool, and a pattern found for checkout can also be shown to staff as evidence about product relationships.

The deterministic seeding process raises confidence in the implementation logic. The test data plants a high probability of a beverage with Kottu or Fried Rice, a seafood side with Rice and Curry, a dessert with a main dish, and Fresh Lime Juice with grilled or devilled seafood. With the random seed fixed, repeated runs use the same synthetic evidence, so rediscovering these patterns is a controlled test of whether mining and ranking behave as intended. Table 7.1 lists the main findings and their significance.

Table 7.1: Interpretation of the principal findings

Finding	Interpretation	Research significance
Hybrid signal fusion	Different signals compensate for one another when data is sparse or uneven.	Reduces dependence on a single recommender technique.
Stock-aware candidate filtering	Only items that can actually be sold are eligible.	Connects analytical relevance with POS operational reality.
Explainable reason text	Users can understand the main basis of a suggestion.	Supports transparency and practical trust.
Customer and staff use of one model	The same patterns support checkout and business insight.	Improves reuse and product integration.

Finding	Interpretation	Research significance
Deterministic planted-pattern validation	Known relationships give a reproducible correctness check.	Strengthens implementation-level validation.

7.2 Comparison with Literature Baselines and Existing Solutions

Chapter 2 identified several broad recommendation families relevant to retail: popularity-based recommendation, association-rule and market-basket analysis, item-based collaborative filtering, and more complex hybrid or learning-based approaches. The Smart Order Recommendation System sits between these. It keeps the interpretability and modest computing needs of market-basket methods and adds item-item similarity and operational priors to reduce the limits of a pure association-rule model.

Against a popularity-only baseline, the approach is more sensitive to context. Popularity is a useful general prior, particularly when the cart is empty or stronger relationships are missing, but it cannot explain why a particular product complements the items already chosen. The system gives more influence to matching association rules and item similarity, and popularity carries a lower weight as a weaker signal and tie-breaker.

Against an association-rule-only baseline, the system can use weaker co-occurrence structure through cosine similarity and can add category and customer-history evidence. Association rules remain central because they provide support, confidence and lift and because their antecedent-to-consequent form maps naturally onto cross-selling. The design does not assume, though, that every useful relationship will meet the minimum support an Apriori rule needs.

Against conventional collaborative filtering, the system needs much less user identity. Anonymous QR customers do not have to create an account, and the only optional personal signal is a previously entered mobile number used to find the products most often ordered at the branch. This limits how deep personalisation can go, but it fits the aim of low-friction checkout and avoids dependence on a dense user-by-item history that small businesses rarely have.

The earlier paper argues that enterprise recommendation systems can be expensive to compute, heavy on infrastructure and a poor match for micro-enterprise conditions. The current implementation answers that concern with a modular Node.js, MongoDB and FastAPI architecture and with interpretable analytical techniques in place of a large deep-learning recommender. It still pays an efficiency cost, because a year of baskets is mined again on every request. The lightweight goal is supported at the architectural level, but efficiency at production scale needs further work such as caching or scheduled precomputation.

The earlier paper also reports numerical precision, recall, latency and business-impact results for an earlier stage of the system. I do not treat these values as baseline results for the current hybrid code, because the present documentation says that no hold-out Precision@K or Hit Rate is calculated and that accepted recommendations are not saved. A cautious comparison therefore looks at method characteristics and demonstrable behaviour, and does not move numerical claims between different development stages. Table 7.2 summarises the comparison.

Table 7.2: Comparison with common recommendation approaches

Approach	Strength	Limitation in this context	Position of the implemented system
Popularity-based	Simple and dependable with little data	Generic and weakly related to the current cart	Used only as a lower-weight supporting signal
Association rules / Apriori	Interpretable co-purchase patterns using support, confidence and lift	Can miss infrequent but meaningful relationships	Retained as the strongest market-basket signal
Item-item collaborative filtering	Captures similarity from shared basket occurrence	Depends on enough item co-occurrence	Added through cosine similarity as a complementary signal
Account-based personalised recommender	Can provide rich individual preferences	Requires user identity and enough history	Replaced by optional phone-based favourites for low-friction QR ordering
Complex deep-learning hybrid	Can model nonlinear and contextual behaviour	Greater data, tuning and infrastructure needs	Not selected for the current micro-enterprise-focused implementation

7.3 Research Contribution

The principal research contribution is a hybrid recommendation strategy designed for the constraints of a micro-enterprise POS environment. I do not propose a new basic recommendation algorithm. The contribution is an applied combination of market-basket rules, item similarity, category-level behaviour, popularity and limited personal history, fused into a transparent ranking that knows about the current cart and branch conditions.

A second contribution is normalised multi-signal score fusion. Lift can be much larger than similarity or attach-rate values, so the implementation transforms it as $\text{lift}/(1 + \text{lift})$ and combines it with confidence before the rule contribution enters the final score. The unbounded scale of lift therefore cannot automatically overwhelm other evidence. The strongest contribution for a candidate decides the explanation shown to the user, which links score construction and interpretability.

A third contribution is category-gap reasoning. The system sends the categories already in the cart and can promote popular items from categories that are missing, which gives the recommender a simple awareness of whether the basket is complete. In a restaurant this resembles a staff member noticing that a customer has chosen a main dish but no beverage or dessert.

The work also contributes a low-friction personalisation strategy. Returning QR customers can be recognised through an optional mobile number, and the backend finds their most frequently ordered products at the same branch. The recommender uses these favourites as one signal without any account creation. It is a deliberately limited form of personalisation, shaped by the realities of quick-service ordering.

Finally, the deterministic synthetic dataset is a reproducible validation artefact. The seeding script creates 12,000 orders over a configurable period with known patterns and a fixed random seed, so the expected relationships can be inspected in advance and the pipeline tested against controlled ground truth. It does not replace real-world evaluation, but failures to rediscover a pattern can be investigated deterministically, which is valuable at implementation level. Table 7.3 lists the contributions.

Table 7.3: Research contributions of the Smart Order Recommendation System

Contribution	Description	Value
Hybrid market-basket recommender	Fuses association rules, cosine similarity, attach rates, popularity and favourites.	Balances relevance, coverage and interpretability.
Normalised score fusion	Scales heterogeneous signals before the weighted combination.	Prevents one metric scale from dominating the ranking.
Category-gap reasoning	Uses the categories already in the cart to encourage complementary categories.	Introduces awareness of basket completeness.
Low-friction personalisation	Uses optional phone-based favourites without a login.	Supports returning customers while keeping the QR flow simple.
Deterministic validation artefact	Synthetic planted patterns generated with a fixed seed.	Enables reproducible implementation verification.

7.4 Product Contribution

The product contribution is the integration of recommendation intelligence into the SmartCost POS workflow, not a separate analytical dashboard. The customer-facing Next.js ordering interface can request recommendations at the review stage, show ranked cards with reasons and let suggested products be added before the order is submitted. This puts the recommender at the point where a cross-selling decision can be acted on.

The Node.js backend acts as an orchestration and safety boundary between the operational POS data and the analytical service. It validates the branch and cart, retrieves historical baskets, candidate products, stock state and optional favourites, forwards a structured request to FastAPI, and adds product names, prices, images and readable explanations to the result. The Python layer can concentrate on analysis while the POS backend stays authoritative for business data and access control.

The staff mobile application adds a second use. Staff can inspect model statistics, high-lift buy-together rules and category attach rates in addition to individual recommendations. This turns recommendation data into an operational insight that may help with menu placement, promotions and staff understanding of which products customers combine.

The system is also branch-aware. Historical baskets, stock availability and customer favourites are all prepared for a specific shop and branch. That has commercial weight, since availability and local demand can differ between branches, and a recommendation produced for one branch is not blindly reused at another.

The QR ordering module is functionally implemented and validated in the development and testing environment but has not been publicly hosted. The product contribution should be read as an integrated, working development implementation and not as a completed public production deployment. I keep this distinction throughout the thesis so as not to overstate operational maturity.

7.5 Limitations and Threats to Validity

The most important validity limitation is that no offline recommendation-accuracy experiment exists for the current hybrid implementation. The code does not yet hide known items from historical baskets and calculate Precision@K, Recall@K or Hit Rate. The thesis can therefore show deterministic pattern rediscovery and inspect rule statistics, but it cannot claim a full quantitative estimate of top-N accuracy for the current model.

A related limitation is that accepted recommendations are sent from the QR interface but are not stored by the backend. Real uptake rate, incremental average order value and revenue attributable to recommendations cannot be measured from the present system. This is a construct-validity issue, because the business goal of cross-selling depends on how customers respond and not only on whether the system can generate plausible suggestions.

Synthetic validation threatens internal and external validity as well. Planted patterns are useful because their expected relationships are known, but synthetic behaviour is simpler than live customer behaviour. Real restaurants have seasonality, promotions, substitutions, menu changes, price sensitivity and irregular ordering that the generated transactions do not fully capture. Recovering planted patterns therefore shows algorithmic correctness under controlled conditions and does not guarantee production effectiveness. Figure 7.2 shows the main threats and the mitigation directions.

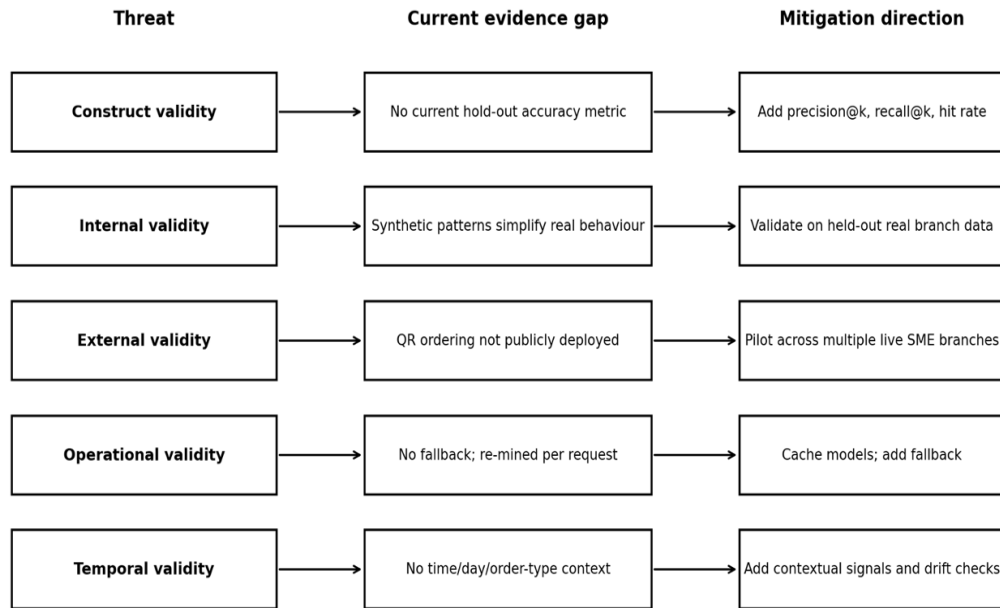


Figure 7.2: Main threats to validity and corresponding mitigation directions

The fixed minimum support count is another limitation. A threshold of three baskets may be permissive for a high-volume shop and restrictive for a low-volume or newly opened branch. New products with fewer than three historical occurrences also suffer from cold start and may not appear through the learned signals. A support threshold that adapts to branch volume would generalise better across businesses of different sizes.

The five recommendation weights are hand-tuned design choices and not parameters learned from observed acceptance behaviour. This brings researcher and design bias into the ranking function. The scheme is transparent and reproducible, but it cannot be claimed to be optimal. Storing accepted recommendations would allow later optimisation or learning-to-rank approaches to estimate the weights from real user responses.

The model also lacks temporal and situational context. Time of day, day of week and order type are not used. A beverage associated with lunch may not suit breakfast equally, and dine-in behaviour may differ from takeaway. Without this context the system adapts less well to changing purchase conditions.

There are architecture-level limitations too. A full year of baskets is mined again on each request, so response cost can rise with transaction volume. There is no fallback recommendation engine if the Python analysis service is unavailable. And because the QR ordering feature is not yet deployed publicly, the implementation has not met sustained external traffic, mobile network variability or real production concurrency. Table 7.4 summarises the threats and the improvement each one calls for.

Table 7.4: Limitations and threats to validity

Limitation / threat	Effect on interpretation	Required improvement
No current hold-out accuracy metric	Current top-N accuracy cannot be quantified.	Implement Precision@K, Recall@K and Hit Rate.
Accepted recommendations not stored	Uptake and incremental revenue cannot be measured.	Store recommendation exposure and acceptance events.
Synthetic planted-pattern validation	Controlled data may not capture real behavioural complexity.	Evaluate with held-out and live branch data.
Fixed support threshold	May not generalise across branch volumes.	Use a volume-relative or adaptive support.
Cold start for new products	New items have too little historical evidence.	Add content or category priors, or exploration logic.
Hand-tuned signal weights	Ranking weights may not be optimal.	Learn or tune weights from measured outcomes.
No temporal or order-type context	Recommendations may ignore situational relevance.	Add contextual features.
Re-mining on each request / no fallback	Scalability and resilience are limited.	Cache models and provide fallback recommendations.
No public QR deployment	Production behaviour is not yet established.	Run a controlled live deployment with monitoring.

7.6 Implications

For micro-enterprise operators, the work implies that useful recommendation does not have to depend on a large customer-account ecosystem or an opaque enterprise-scale model. Transaction histories that a POS already generates can supply enough structure to find co-purchase relationships, and branch inventory and cart context make those relationships actionable. This lowers the barrier to adopting data-driven cross-selling in smaller businesses.

For system designers, the results stress the need to join machine-learning or data-mining logic with operational constraints. A recommender should not be judged on

statistical association alone. It must respect stock, branch identity, cart contents and the application workflow. The pattern shown here is that the operational backend prepares trusted business context, the analytical service ranks, and the user interface receives a result already enriched with an explanation.

For responsible and explainable AI practice, the system suggests that fairly simple explanation mechanisms can be valuable when they are tied directly to the scoring process. Showing why an item was suggested makes the recommendation inspectable by staff and understandable by customers. The approach does not remove every concern about personalisation or data use, especially where a mobile number is a repeat-customer signal, but the optional and limited nature of that signal reduces the need for extensive profile data.

For future research, the most important implication is that the next stage should move from functional correctness to measuring outcomes. The system should record recommendation exposures and accepted items, add an offline leave-one-out or hold-out evaluation, and then compare alternative weights or ranking variants. That would test whether the hybrid fusion really improves relevance compared with association rules or popularity alone.

For production engineering, caching and scheduled model refresh are the high-priority improvements. Mining per request is acceptable for development validation but becomes inefficient as order volume grows. Precomputing frequent itemsets, similarity neighbours and category statistics for each branch, and refreshing them on a schedule or after a set number of new transactions, would improve scalability while keeping the current API design.

A public pilot deployment would also supply evidence that is missing today. Real operation could measure latency under network conditions, recommendation acceptance, changes in average order value, differences between branches and user feedback. Since the QR ordering interface is already functionally integrated in development, a pilot is a natural next step and not a redesign of the core recommendation architecture.

In summary, the discussion supports the feasibility and practical relevance of the Smart Order Recommendation System and also defines the evidence still needed before strong production-performance claims can be made. The hybrid approach is a practical bridge between market-basket analytics and day-to-day POS operation, but its full research and commercial value depends on real-world deployment, systematic outcome logging, comparative offline evaluation and engineering for scale and resilience.

8 COMMERCIALISATION AND SUSTAINABILITY

This chapter looks at how the Smart Order Recommendation System could create practical value beyond the research prototype. The discussion rests on what the implemented system actually shows: integration with a POS workflow, use of existing transaction history, lightweight analytical services, explainable recommendations, branch-aware stock filtering, and one recommendation model serving both customer suggestions and staff insights. I do not treat unverified revenue figures or pricing assumptions as established results. The chapter identifies commercially relevant capabilities and a realistic path from the current development-stage implementation to an operational product for small businesses.

8.1 Value Proposition and Target Users

The main value proposition is turning routine POS transaction data into real-time help with selling. Typical small-business POS platforms record transactions, manage billing and maintain inventory, but they do not necessarily use the order history to guide the next purchase decision. The Smart Order Recommendation System fills that gap by producing ranked complementary-product suggestions when an order is reviewed, and by showing staff the buying patterns behind them. It connects historical data to an immediate decision: which additional item is most appropriate to suggest now.

The primary target users are micro and small businesses that already generate transaction data but have no data-science team and no enterprise recommendation infrastructure. Bakeries, cafés, restaurants and small retail shops suit this well, since they often sell complementary products and depend on fast checkout. Association-based recommendations are a good fit there because combinations such as a main item with a beverage, side or dessert can be learned from baskets without detailed customer profiles [3], [10], [11].

A second target group is operational staff and business owners. Customer-facing recommendations support cross-selling at checkout, while the staff view exposes model statistics, high-lift buy-together rules and category attach rates. The same

analytical pipeline therefore serves both a transaction-level action and management insight. The owner is not left with a black-box output and can see, for example, which categories often appear with main items and which products are strong consequents in association rules.

The system is also designed to cause little friction for QR-ordering customers. The recommendation request is public in the customer ordering flow, and optional mobile-number history can identify frequently bought items without a login. This gives light personalisation and keeps checkout simple. The QR interface is functionally implemented and validated in the development environment, although it has not yet been deployed as a public production service. Table 8.1 summarises the value for each stakeholder.

Table 8.1: Commercial value proposition by stakeholder

Stakeholder	Operational need	System value
Customer	Fast ordering with relevant add-on choices	Cart-aware, explainable suggestions before order submission.
Cashier / staff	Consistent cross-selling support	Buy-together rules and clear recommendation reasons.
Owner / manager	Actionable use of POS history	Model statistics, attach rates and product-pair insights.
Micro-enterprise	Low-complexity intelligence	Modular architecture that uses existing transactional data.

8.2 Market and Competitive Position

The market position of the system comes less from replacing full enterprise recommender platforms and more from building a focused recommendation capability into a POS environment made for small businesses. The literature reviewed in Chapter 2 indicates that many advanced recommenders depend on large user histories, specialised infrastructure or computationally heavy models. Association-rule mining and item-based techniques, by contrast, can extract useful relationships from transactional data using fairly interpretable logic and manageable computing needs [3]–[5], [11].

The implemented system also differs from a basic popularity list. Popularity is kept only as one weaker signal, and stronger contextual evidence can come from matching

association rules, item-item similarity, a customer favourite or a category gap in the current basket. This has commercial importance, because a generic “popular items” panel is easy to build but does not react to what the customer has already chosen. The hybrid approach aims for better relevance without a high-complexity deep-learning stack. Table 7.2 compared the approaches in Chapter 7.

Explainability is a further competitive attribute. Each recommendation carries a reason code that is translated into customer-facing language, for example that an item is often ordered with something in the cart, is a previous favourite, is popular in a missing category or is popular overall. Small businesses are more likely to adopt a tool whose output non-technical staff can understand, since opaque recommendations create a trust barrier [1], [11].

The system should nevertheless be described as a research-backed prototype and not as a finished commercial product. It is not publicly deployed, does not store recommendation acceptance events and mines historical baskets again on each request. These characteristics limit current production scalability and prevent direct measurement of incremental revenue or recommendation uptake. Commercial claims should therefore rest on the implemented capability and the demonstrated end-to-end workflow, not on promises of sales growth that nothing supports.

8.3 Business or Sustainability Model

A sustainable business model should start from the fact that the recommendation capability is not a standalone consumer application. It is a feature that gains its value from the wider SmartCost POS environment. A practical commercial route is to bundle the recommendations into the POS service or to offer them as an enhanced analytics tier. The engine would then reuse the transaction, product, category, branch and stock data that the POS already holds, and avoid the cost and friction of a separate data-integration project.

From an operational-sustainability view, the split between the Node.js transactional backend and the Python FastAPI analysis service is helpful. The analytical component can change independently while core transaction processing stays stable. The use of MongoDB, REST-based communication, sparse matrices and conventional data-

mining methods also helps maintainability, because the implementation depends on widely used technologies and not on a proprietary recommendation platform [14]–[17].

The present architecture needs optimising before large-scale commercial operation. Mining one year of baskets for every request will raise response time and compute cost as order volume grows. The documented future direction is to cache frequent rules and similarity structures for each branch and refresh them periodically. That would move the expensive mining out of the customer request path and make operating cost more predictable.

Sustainability also depends on measurable business value. The QR page can send accepted recommendation information, but the backend does not store it. A commercial version should record recommendation impressions, additions, removals and the resulting order values, so that each shop can see acceptance rate, incremental order value and category-level performance. Without that feedback loop the feature can be demonstrated technically but cannot be managed as a commercial capability that keeps improving.

8.4 Scalability, Intellectual Property and Compliance

Scalability is supported in principle by branch-specific processing. Recommendations are generated from the shop and branch context, a 365-day order window, branch stock and branch-local customer history. This lowers the risk of applying patterns from one branch to another by mistake and gives a natural unit for future caching, scheduled retraining and horizontal scaling. The public customer API and the protected staff API also keep anonymous ordering apart from authenticated management access.

The main scalability constraint is request-time mining. The engine computes frequent itemsets, rules, item similarities, category attach rates and ranking signals from the baskets supplied with each request. That suits a controlled prototype, but production scale would call for precomputed artefacts, branch-level caches, incremental updates, monitoring and explicit service-level performance targets. I list these as future engineering work and do not claim them as completed features.

On intellectual property, the protectable value lies mainly in the integrated workflow and the implementation know-how: branch-aware basket preparation, stock-aware candidate filtering, hybrid score fusion, explainable reason selection, diversity handling and the reuse of the model for customer and staff views. Standard algorithms such as Apriori and cosine similarity are established techniques and should not be presented as proprietary inventions [3]–[5]. Any commercial intellectual property (IP) claim should focus on the system-specific orchestration, the software implementation, the interface design and the resulting product configuration, not on ownership of the underlying algorithms.

Compliance covers privacy, access control, secure handling of customer identifiers and transparent use of recommendation logic. The system uses an optional mobile number as a personalisation signal, so a production deployment should keep retention short, restrict access, define lawful and transparent use, and avoid exposing customer-level history in staff analytics beyond what operations need. Staff endpoints are authenticated, while customer recommendation requests are public but validated against shop and branch context. A production deployment should add formal privacy notices, logging, rate limiting, secure transport, secret management and a documented data-retention policy. Table 8.2 lists the priorities.

Table 8.2: Scalability and compliance priorities

Area	Current design	Commercialisation requirement
Computation	Mining performed during requests	Cache or precompute per branch and refresh on a schedule.
Data isolation	Shop and branch context	Formal tenant-isolation tests and access controls.
Customer data	Optional phone-based favourites	Minimisation, a retention policy and a privacy notice.
Staff access	JWT-protected insights endpoint	Role-based permissions and audit logging.
Reliability	A Python service failure returns an error	Fallback and retry strategy, plus monitoring.
Measurement	Acceptance events not stored	Store impressions and acceptance, and report business Key Performance Indicators (KPIs).

8.5 Contribution to the Sustainable Development Goals

The contribution to the United Nations Sustainable Development Goals (SDGs) should be read as an enabling contribution and not as a measured impact. The system is built for micro and small enterprises, and its main sustainability relevance is the use of digital decision support to make better use of data a business already has. This fits most closely with SDG 8 (Decent Work and Economic Growth) and SDG 9 (Industry, Innovation and Infrastructure), because it shows how accessible analytics can be built into everyday small-business operations.

Under SDG 8, the system may help small businesses make more consistent sales decisions and get more value from transaction data without specialist analytics staff. Under SDG 9, the project offers a light digital-innovation architecture that joins mobile and web interfaces, backend services, a database and explainable recommendation analytics. The emphasis on modular, widely adopted technologies supports practical digitalisation and does not depend on specialised infrastructure.

There is also a secondary link to SDG 12 (Responsible Consumption and Production). The candidate set checks branch-level availability and excludes unavailable items, so recommendations are tied to real inventory. Future work could strengthen the link by combining recommendation logic with expiry, waste, stock-age or overstock signals, so that add-on suggestions help cut avoidable food and product waste and do not simply maximise transaction value. Table 8.3 gives the indicative alignment.

The SDG discussion above is an analytical mapping of the project's capabilities. The sources used in this research include no measured SDG impact study, so no quantified sustainability outcome is claimed.

Table 8.3: Indicative SDG alignment

SDG	Project linkage	Evidence level
SDG 8 – Decent Work and Economic Growth	Decision support and sales intelligence for micro and small enterprises.	Conceptual / enabling contribution
SDG 9 – Industry, Innovation and Infrastructure	Lightweight digital recommendation architecture integrated with POS.	Direct project design contribution
SDG 12 – Responsible Consumption and Production	Stock-aware candidate filtering; potential future waste-aware recommendations.	Partial current linkage; stronger future opportunity

9 CONCLUSION AND FUTURE DIRECTIONS

9.1 Conclusion

This thesis looked at the design, implementation and validation of a Smart Order Recommendation System for POS environments used by small businesses. The practical problem was that small businesses routinely collect transaction histories but mostly use them for record keeping and not for real-time decision support. The component I built turns completed order baskets into suggestions that can be shown during QR checkout and into insights that staff can read.

The final implementation is a hybrid market-basket recommender and not a single-technique association-rule module. It combines Apriori frequent-itemset mining and association rules with item-item cosine similarity, category attach rates, customer favourites and overall popularity. The signals are normalised and fused through weighted scoring, and a diversity rule then keeps the list from being dominated by a single reason. Only orderable, in-stock candidates outside the current cart are eligible, and the dominant signal is translated into a readable explanation.

The component is integrated across a Next.js customer ordering interface, Node.js/Express orchestration, MongoDB transactional data, a Python/FastAPI analysis service and an Expo/React Native staff interface. This shows that recommendation logic can sit inside the operational flow of a POS system and need not stay an offline analytical experiment. The customer QR workflow is functionally implemented and tested in the development environment, and public production hosting remains future deployment work.

Validation of the hybrid implementation is strongest at the functional and pattern-recovery levels. The deterministic seed process generates 12,000 synthetic orders with planted relationships, so the recommender can be checked against known co-purchase patterns. The documentation also records model statistics, rule outputs, category attach rates and end-to-end API behaviour. The codebase does not yet implement a hold-out Precision@K or Hit Rate evaluation, though, and accepted recommendations are not stored for calculating real uptake or incremental revenue. I treat these gaps as future evaluation requirements and have not hidden them behind unsupported claims.

9.2 Achievement of Objectives

The primary objective, an integrated and explainable recommendation capability for small-business POS use, has been achieved at prototype level. Historical branch transactions can be turned into basket data, mined for frequent relationships, enriched with similarity and contextual signals, and returned as ranked customer recommendations. The same process can be run with an empty cart to give staff-level insights, so the recommendation core is reused in two user contexts.

Cart relevance has been addressed through rule matching, item similarity and explicit cart-category context. Operational relevance has been addressed through stock-aware candidate generation and branch-specific filtering. Explainability comes from reason codes and plain-language messages. Light personalisation is provided by optional mobile-number favourites, with no user accounts. Variety is handled by a diversity rule that can replace the final ranked item when all top recommendations share one reason.

Reproducible validation has been achieved only in part, through deterministic synthetic data. The seed script uses a fixed random seed and known planted patterns, so repeated experiments are consistent and the system can be checked for recovery of the intended product relationships. A full recommender accuracy evaluation, controlled A/B testing, production latency benchmarking under load and measurement of commercial uplift are incomplete and are carried forward as future directions. Table 9.1 shows the status of each objective area.

Table 9.1: Objective achievement summary

Objective	Achievement status	Evidence
Generate contextual recommendations from POS history	Achieved at prototype level	Hybrid recommendations returned from branch baskets and cart context.
Support explainable recommendations	Achieved	Reason codes and readable reason sentences.
Avoid unavailable recommendations	Achieved	Branch stock and orderability filtering.
Provide staff-level recommendation insights	Achieved	Rules, attach rates and model statistics.

Objective	Achievement status	Evidence
Validate reproducibly	Partially achieved	Deterministic synthetic planted-pattern dataset.
Measure offline ranking accuracy	Not yet completed	No current Precision@K or Hit Rate hold-out evaluation.
Measure commercial uplift	Not yet completed	Accepted recommendations are not stored.

9.3 Individual Research and Product Contributions

My individual research contribution is a practical hybrid recommendation approach adapted to the constraints of micro-enterprise POS environments. The component does not rely on a single association rule. It combines complementary evidence sources that cover different weaknesses: Apriori captures strong co-purchase rules, cosine similarity captures weaker item relationships, customer favourites add light personalisation, category attach rates fill gaps in the basket, and popularity provides a controlled fallback signal.

A further research contribution is the normalised score-fusion strategy. Association-rule lift is transformed into a bounded value before being combined with confidence, so unbounded lift values cannot overwhelm signals whose natural range lies between zero and one. The system uses only the strongest matching rule for each candidate and does not accumulate many weak ones, and a deterministic product ID serves as tie-breaker. These decisions make the ranking more stable and easier to interpret.

The product contribution is the end-to-end integration of recommendation intelligence into a working POS ecosystem. It covers data preparation from completed orders, candidate filtering against the live catalogue and branch stock, orchestration through Node.js, model execution through FastAPI, the customer-facing “Add something else?” interaction and the staff-facing model insights. The result is more than an algorithmic notebook. It is an integrated software component with operational interfaces and explicit API contracts.

Another contribution is the validation artefact itself. The deterministic seed generator creates realistic order baskets with known attach probabilities and controlled noise. Because the planted relationships are known in advance, the test data gives a repeatable

check that the recommender rediscovers meaningful patterns. This is especially valuable when real commercial transaction data is scarce or cannot be shared freely for research.

9.4 Future Directions

The first priority is rigorous recommendation evaluation. The current implementation should be extended with a leave-one-out or hidden-item protocol in which one item is removed from historical baskets and the recommender is tested on whether the hidden item appears in the top-k output. Precision@K, Recall@K, Hit Rate, mean reciprocal rank and coverage can then be calculated under reproducible conditions, which would strengthen the empirical evidence of recommendation quality directly.

The second priority is closed-loop outcome measurement. Recommendation impressions, accepted suggestions, rejected suggestions and final order values should be stored with suitable privacy controls. These events would let the system report acceptance rate, incremental average order value, category-level uplift and recommendation-attributed revenue. In time, the same data could be used to learn the signal weights and replace the manually chosen ones.

Scalability should be improved by caching or precomputing the frequent rules, similarity neighbours, category attach rates and popularity statistics for each branch. These artefacts can be refreshed on a schedule or incrementally as new orders arrive. Moving mining out of the customer request path would reduce latency variation and suit larger transaction volumes better. A graceful fallback should also be added, so that a temporary failure of the analytics service does not make the customer recommendation endpoint fail completely.

Recommendation context can be widened beyond the current cart. Time of day, day of week, order type, season, promotion state and customer segment may all affect how suitable a product is. These signals should be added only after controlled evaluation, because greater model complexity does not automatically make the system more useful commercially. Context-aware features should keep the project's emphasis on explainability and operational simplicity.

Cold-start handling also needs work. New products currently have too little evidence until they meet the minimum support requirement. Later versions could use category priors, content attributes, manually curated complementary relationships or exploration policies so that new products get exposure without weakening established rules. The minimum support threshold itself should adapt to order volume and not stay fixed at three baskets for every branch.

Finally, the QR ordering module should be deployed to a production hosting environment and evaluated with real business traffic. Production deployment would allow load testing, monitoring, security hardening, privacy validation and controlled field experiments. That stage would provide the evidence needed to move from a technically working research prototype to a commercially defensible recommendation feature.

REFERENCES

- [1] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Amsterdam, The Netherlands: Elsevier, 2012.
- [2] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Cambridge, MA, USA: Morgan Kaufmann, 2016.
- [3] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, 1994, pp. 487–499.
- [4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proc. 10th Int. World Wide Web Conf.*, 2001, pp. 285–295.
- [5] M. Deshpande and G. Karypis, “Item-based top-N recommendation algorithms,” *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, 2004.
- [6] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, Jun. 2005.
- [7] D. J. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. Cambridge, MA, USA: MIT Press, 2001.
- [8] C. C. Aggarwal, *Data Mining: The Textbook*. Cham, Switzerland: Springer, 2015.
- [9] P. Resnick and H. R. Varian, “Recommender systems,” *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.
- [10] J. B. Schafer, J. Konstan, and J. Riedl, “E-commerce recommendation applications,” *Data Mining and Knowledge Discovery*, vol. 5, no. 1–2, pp. 115–153, 2001.
- [11] F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*, 3rd ed. New York, NY, USA: Springer, 2022.
- [12] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.
- [13] A. McAfee and E. Brynjolfsson, “Big data: The management revolution,” *Harvard Business Review*, vol. 90, no. 10, pp. 60–68, 2012.
- [14] OpenJS Foundation, “Node.js documentation,” 2025. [Online]. Available: <https://nodejs.org/en/docs/>

- [15] MongoDB Inc., “MongoDB documentation,” 2025. [Online]. Available: <https://www.mongodb.com/docs/>
- [16] FastAPI, “FastAPI framework documentation,” 2025. [Online]. Available: <https://fastapi.tiangolo.com/>
- [17] React Native, “React Native documentation,” 2025. [Online]. Available: <https://reactnative.dev/docs/getting-started>
- [18] Sri Lanka Institute of Information Technology, *Guidelines on Documentation and Submission of Dissertations*, 2025.
- [19] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User Model. User-Adapt. Interact.*, vol. 12, no. 4, pp. 331–370, 2002. doi: 10.1023/A:1021240730564.
- [20] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004. doi: 10.1145/963770.963772.
- [21] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, “Improving recommendation lists through topic diversification,” in *Proc. 14th Int. Conf. World Wide Web (WWW)*, 2005, pp. 22–32. doi: 10.1145/1060745.1060754.
- [22] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, “BPR: Bayesian personalized ranking from implicit feedback,” in *Proc. 25th Conf. Uncertainty in Artificial Intelligence (UAI)*, 2009, pp. 452–461. [Online]. Available: <https://arxiv.org/abs/1205.2618>
- [23] T. Schnabel, A. Swaminathan, A. Singh, N. Chandak, and T. Joachims, “Recommendations as treatments: Debiasing learning and evaluation,” in *Proc. 33rd Int. Conf. Machine Learning (ICML)*, vol. 48, 2016, pp. 1670–1679. [Online]. Available: <https://proceedings.mlr.press/v48/schnabel16.html>
- [24] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, 2003. doi: 10.1109/MIC.2003.1167344.

- [25] G. Karypis, “Evaluation of item-based top-N recommendation algorithms,” in Proc. 10th Int. Conf. Information and Knowledge Management (CIKM), 2001, pp. 247–254. doi: 10.1145/502624.502627.
- [26] S. M. McNee, J. Riedl, and J. A. Konstan, “Being accurate is not enough: How accuracy metrics have hurt recommender systems,” in CHI Extended Abstracts on Human Factors in Computing Systems, 2006. doi: 10.1145/1125451.1125659.
- [27] M. Kaminskis and D. Bridge, “Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems,” ACM Trans. Interact. Intell. Syst., vol. 7, no. 1, art. 2, 2016. doi: 10.1145/2926720.
- [28] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, “Methods and metrics for cold-start recommendations,” in Proc. 25th Annu. Int. ACM SIGIR Conf. Research and Development in Information Retrieval, 2002. doi: 10.1145/564376.564421.
- [29] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” Computer, vol. 42, no. 8, pp. 30–37, 2009. doi: 10.1109/MC.2009.263.
- [30] ISO/IEC, ISO/IEC 25010:2023, Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Product quality model, 2023. [Online]. Available: <https://www.iso.org/standard/78176.html>
- [31] National Institute of Standards and Technology, Security and Privacy Controls for Information Systems and Organizations, NIST SP 800-53 Rev. 5, 2020. doi: 10.6028/NIST.SP.800-53r5.
- [32] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?: Explaining the predictions of any classifier,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2016, pp. 1135–1144. doi: 10.1145/2939672.2939778.

- [33] Y. Zhang and X. Chen, “Explainable recommendation: A survey and new perspectives,” *Found. Trends Inf. Retr.*, vol. 14, no. 1, pp. 1–101, 2020. doi: 10.1561/15000000066.
- [34] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, “Session-based recommendations with recurrent neural networks,” in *Proc. 4th Int. Conf. Learning Representations (ICLR)*, 2016. [Online]. Available: <https://arxiv.org/abs/1511.06939>
- [35] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *Proc. 26th Int. Conf. World Wide Web*, 2017, pp. 173–182. doi: 10.1145/3038912.3052569.

GLOSSARY

Apriori – A frequent-itemset mining algorithm that expands itemsets level by level and prunes candidates whose subsets are not frequent.

Association rule – An implication of the form $A \rightarrow B$ that describes how often product B appears in transactions containing product set A.

Attach rate – The proportion of baskets containing at least one product from a specified category.

Candidate product – An orderable product that is eligible to be scored for recommendation after cart and stock filtering.

Cold start – The difficulty of recommending a new or rarely ordered product that has too little interaction history.

Confidence – For a rule $A \rightarrow B$, the proportion of baskets containing A that also contain B.

Cosine similarity – A similarity measure used in the system to compare binary item-occurrence vectors across order baskets.

Coverage – The share of items that the recommendation model can reach through rule consequents or similar-item neighbours.

Diversity rule – A post-ranking rule that brings in a different recommendation reason when the top list is dominated by one signal type.

Frequent itemset – A group of products that occurs together in at least the configured minimum number of order baskets.

Hybrid recommender – A recommendation approach that combines several signals or algorithms and does not depend on one technique.

Item-item collaborative filtering – A technique that recommends products based on the similarity between product interaction patterns.

Lift – Association-rule strength relative to chance; values above one indicate a positive relationship.

Market basket analysis – Analysis of transaction baskets to discover products that frequently occur together.

modelReady – A status flag showing whether enough historical baskets exist to generate recommendation outputs.

Precision@K – The proportion of the top-K recommended items that are relevant under a defined evaluation protocol.

QR ordering – A web-based customer ordering workflow entered by scanning a restaurant or table QR code.

Recommendation reason – A readable explanation generated from the strongest contributing ranking signal.

Support – The proportion of all baskets that contain the complete itemset represented by an association rule.

Top-N recommendation – A ranked list containing the N highest-scoring candidate products.

Transactional basket – The distinct set of products contained in one completed order and used as a mining unit.

APPENDICES

The appendices hold supplementary material that supports reproducibility, validation and transparent reporting of the Smart Order Recommendation System. In line with the dissertation guidelines [18], detailed technical material is placed here when it would interrupt the main narrative.

Appendix A: Research Instruments or Extended Results

The project sources do not include the original wording of a formal questionnaire or interview schedule, so this appendix does not invent a research instrument. It records the validation artefacts, experimental parameters and evaluation dimensions that are explicitly documented for the recommendation component.

A.1 Deterministic seed parameters and planted relationships

Table A.1 gives the parameters of the deterministic recommendation seed. The planted ground-truth relationships are those already listed in Table 5.4. The purpose of the planted-pattern design is to create a known answer against which the recommender can be inspected. A correct implementation should recover the strong injected relationships as frequent pairs or high-lift rules despite the controlled noise. The fixed random seed makes repeated runs deterministic, which helps regression testing and reproducibility.

Table A.1: Deterministic recommendation seed parameters

Parameter	Documented value
Generated order volume	12,000 orders over 180 days (configurable)
Default target	Shop SI000001, branch B00001
Service-hour distribution	Lunch 11:00–14:00 (55%); dinner 18:00–21:00 (45%)
Basket opener distribution	Kottu 45%; Rice & Curry 34%; Seafood 13%; drink or dessert only 8%
Noise	One random extra item in 30% of orders; two in 8%
Random seed	20260827
Safety control	Clean mode removes only records carrying the recommendation-seed identifiers; dry-run previews samples

A.2 Core request parameters

Table A.2 lists the parameters of the internal recommendation-service request, with their defaults or meaning.

Table A.2: Recommendation-service request parameters

Parameter	Default / meaning
transactions	Historical order baskets for the branch
cartItemIds	Products currently in the customer cart
cartCategoryIds	Categories already represented in the cart
candidates	Orderable, in-stock products not already in the cart
favouriteIds	Optional products frequently ordered by the supplied phone number
limit	Default 10; customer endpoint constrained to 1–12
minSupportCount	3 baskets
maxAntecedentSize	2 products
topNeighbors	8 similar items
maxRules	200 retained rules

A.3 Documented model outputs

Table A.3 explains the model statistics returned by the service, which are also described in Table 6.2.

Table A.3: Model statistics and interpretation

Statistic	Interpretation
transactionCount	Number of order baskets used by the model
itemCount	Products meeting the minimum support requirement
ruleCount	Retained association rules with lift greater than one
pairCount	Number of frequent product pairs
avgBasketSize	Average number of distinct products per order
coverage	Share of items reachable as rule consequents or similarity neighbours

A.4 Evaluation dimensions reported in the research paper

The group's earlier research paper reports a practical user evaluation with eight bakery and café cashiers and managers. The evaluation dimensions were recommendation usefulness, ease of understanding, checkout usability, perceived improvement in sales opportunity and explainability. The source text does not give the exact questionnaire

statements or the response-scale table, so I list the dimensions without reconstructing item wording that I cannot verify:

- Perceived usefulness of recommendation suggestions during checkout.
- Ease of understanding the recommendation panel and the recommendation reasons.
- Compatibility of the feature with the normal checkout workflow.
- Perceived support for cross-selling or additional-product suggestions.
- Trust in, and interpretability of, the recommendation explanations.

The earlier paper reports quantitative accuracy, latency and business-impact values. The current hybrid code documentation states that no hold-out Precision@K or Hit Rate evaluation is implemented and that accepted recommendations are not stored. The earlier values should therefore be read as results reported in a prior study and not as measurements re-verified from the current codebase.

Appendix B: Academic Integrity and AI Use Disclosure

This appendix records the use of generative AI and other automated tools in producing this thesis, as required by the module policy of the Sri Lanka Institute of Information Technology (SLIIT) and by the declaration at the front of the thesis. Table A.4 lists each tool, what it was used for, how its output was checked and which parts of the thesis it affected. The design, the implementation, the data and the conclusions described in the thesis are my own work, and I remain responsible for their accuracy.

Table A.4: Disclosure of AI and automated tool use

Tool	Purpose	Verification performed	Sections or artefacts affected
Generative AI assistant (Claude, Anthropic)	Restructuring an earlier draft of this thesis into the SLIIT template and rewording the text.	To be completed by the author before submission.	Text of Chapters 1–9, front matter and appendices.
[Other tool, if any]	[Purpose]	[Verification]	[Sections affected]