INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 4, April 2026 | ISSN: 2456-4184 | IJNRD.ORG

DESIGN AND IMPLEMENTATION OF A MICROSERVICES-BASED UNIFIED CRM PLATFORM FOR E-COMMERCE CUSTOMER DATA CONSOLIDATION

S. Ashok Kumar, Divya Dharshan S, S G Srivarshini, Vishnupriya K
Assistant Professor, Student, Student, Student
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering, Sriperumbudur, Tamilnadu, India

Abstract : In the contemporary e-commerce landscape, customer data is frequently distributed across multiple disconnected and heterogeneous systems, preventing staff at all levels from obtaining a unified, real-time view of customer information. This fragmentation results in inconsistent service delivery, poor personalization, and ineffective decision-making. The proposed system addresses this challenge by designing and developing a Unified Customer Relationship Management (CRM) platform built on a microservices architecture. The platform consolidates customer profiles, transaction histories, membership details, loyalty rewards, and behavioral patterns into a single, role-sensitive interface accessible to administrators, store managers, and customer-facing consultants. A fine-grained, role-based access control mechanism governs the visibility and editability of data for each user category. An event-driven integration layer ensures near real-time synchronization of customer data across heterogeneous backend systems, while a distributed in-memory caching layer guarantees low-latency response times during peak operational periods. Scheduled background synchronization tasks handle data ingestion from external systems that cannot be directly connected to the platform. The proposed solution demonstrably enhances customer engagement, improves operational efficiency, reduces service resolution times, and enables data-driven decision-making across the retail enterprise. Performance evaluations confirm that the system meets latency, throughput, and availability targets under realistic load conditions.

Keywords **:** Customer Relationship Management, Microservices Architecture, Role-Based Access Control, Event-Driven Integration, Distributed Caching, E-Commerce, Customer Profile, Data Synchronization, Real-Time Systems, Service-Oriented Architecture, API Gateway, Full Stack Development.

I. INTRODUCTION

The rapid growth of e-commerce has fundamentally transformed the relationship between retail organizations and their customers. In a modern online retail environment, a single customer may interact with an organization across a multitude of channels - browsing a product catalog, placing and tracking orders, redeeming loyalty rewards, contacting customer support via chat or phone, or visiting a physical store for a product trial or pickup. Each of these touchpoints generates data that is valuable for understanding the customer's preferences, purchase intent, and satisfaction levels.

However, in most large-scale e-commerce organizations, this data is not stored in a unified repository. Instead, it is captured by specialized, purpose-built systems that were selected independently over time - an order management system, a loyalty program platform, a customer support ticketing system, a membership and subscription service, and various third-party databases maintained by external partners. The result is a fragmented landscape of customer information that no single team member can access comprehensively.

This fragmentation has measurable consequences for operational performance. Customer-facing consultants are forced to navigate between four or five different applications to assemble a complete picture of a customer before or during an interaction. The time spent context-switching and cross-referencing data reduces the number of customers that can be served per hour and increases the likelihood of errors. Customers who call for support frequently experience the frustration of repeating information they have already provided, because the support agent cannot see the history recorded by a different team in a different system.

At the management level, the absence of a unified data view prevents meaningful analysis of customer segments, purchase patterns, and loyalty program effectiveness. Reports must be manually assembled from multiple exports, a process that is both time-consuming and prone to inconsistencies introduced by differing data schemas and update frequencies across systems.

Customer Relationship Management (CRM) systems have long been proposed as a solution to data fragmentation. Traditional monolithic CRM platforms consolidate customer data into a single application, providing a unified view to all authorized users. However, monolithic systems carry well-documented architectural limitations: a change in any module requires full-application redeployment, performance bottlenecks in one area degrade the entire system, and scaling the platform to meet peak demand requires provisioning resources for the full application even when only a subset of modules is under load.

The microservices architectural pattern addresses these limitations by decomposing a large application into a collection of small, independently deployable services, each responsible for a distinct business capability. Each microservice owns its data store, can be developed and deployed independently of other services, and communicates with peers through well-defined, versioned

interfaces. This approach enables fine-grained horizontal scaling, independent release cycles, and technology heterogeneity - the ability to choose the most appropriate database or framework for each service's specific requirements.

The proposed Unified CRM Application applies the microservices pattern to the problem of customer data consolidation in e-commerce. The system presents a single, integrated customer profile view that aggregates data from multiple backend services in real time, tailored to the role of the authenticated user. An event-driven messaging layer ensures that data updated in any part of the organization is propagated to the CRM with minimal delay. A distributed cache serves as a performance buffer between the frontend and the database layer, and a scheduled synchronization subsystem handles data ingestion from external systems that are not directly accessible.

This paper makes the following contributions. First, it presents a microservices-based architectural design for a unified CRM platform suitable for large-scale e-commerce organizations. Second, it describes a role-based access control model that enforces fine-grained, attribute-level data restrictions across three distinct user roles. Third, it presents an event-driven integration architecture for near real-time data synchronization across heterogeneous systems. Fourth, it provides a detailed account of the implementation methodology, including the development approach, integration strategy, and testing framework. Finally, it reports the results of a performance evaluation conducted under realistic load conditions.

The remainder of this paper is organized as follows. Section II reviews related literature. Section III describes the system architecture. Section IV presents the design methodology. Section V covers the implementation in detail. Section VI presents and discusses the evaluation results. Section VII concludes the paper, and Section VIII outlines directions for future work.

The significance of the customer data unification problem extends beyond operational efficiency. Research in consumer psychology has consistently demonstrated that customers who feel understood by a brand — whose preferences are acknowledged and whose history is respected — exhibit measurably higher loyalty, larger basket sizes, and lower churn propensity than customers who experience impersonal, transactional interactions. The capability to deliver this level of personalized engagement at scale requires the kind of unified customer view that the proposed system provides.

Furthermore, the regulatory environment in which e-commerce organizations operate is increasingly demanding comprehensive records of all customer data stored, how it was obtained, and how it is used — requirements imposed by data protection legislation in multiple jurisdictions. A fragmented data landscape, in which customer information is scattered across systems with varying levels of data governance maturity, makes compliance with these requirements substantially more difficult and more expensive than a centralized, well-governed data platform.

The proposed system addresses both the operational and the regulatory dimensions of the customer data management challenge. By consolidating data into a unified platform with a comprehensive audit log and well-defined data ownership boundaries, it simplifies the implementation of data subject access requests, right-to-erasure compliance, and data retention policy enforcement — all requirements that have become standard obligations for e-commerce organizations operating in regulated markets.

The choice of the microservices architectural pattern is not merely a technical preference but reflects the organizational reality of large e-commerce organizations. In organizations of significant scale, different customer data domains are typically owned by different teams, each with their own development priorities, technology choices, and release cadences. A monolithic architecture requires all teams to coordinate on a shared deployment, creating organizational bottlenecks that slow the evolution of each domain. The microservices pattern aligns the technical architecture with the organizational structure, enabling each team to own and evolve their service independently - a principle articulated in Conway's Law and its corollary, the Inverse Conway Maneuver, which suggests that organizations seeking to achieve a specific software architecture should design their team structures to mirror that architecture.

II. LITERATURE REVIEW

Customer relationship management, microservices architectures, distributed data integration, and event-driven system design are four active areas of research that intersect in the design of the proposed platform. The following studies constitute the key body of literature upon which the system design is grounded.

Customer Relationship Management: Mechanisms and Impact (Soltani & Navimipour, 2016). Soltani and Navimipour conduct a comprehensive systematic review of CRM mechanisms, synthesizing findings from over two hundred peer-reviewed studies. Their analysis identifies three foundational pillars of effective CRM: data centralization, process integration, and analytical capability. They find that organizations which successfully unify customer data across departments consistently outperform fragmented competitors on key metrics including customer retention rate, average order value, and net promoter score. The study also catalogues the most common failure modes of CRM implementations, with data quality degradation and organizational resistance to process change cited as the leading causes of underperformance. However, the review predates the widespread adoption of microservices architectures and does not address the challenges of role-based, attribute-level data access or the real-time cross-system synchronization requirements that characterize modern e-commerce environments.

A Systematic Framework for Successful CRM Integration (Marcinkevage & Kumar, 2025). Marcinkevage and Kumar propose a structured integration framework for CRM deployment, grounded in an analysis of thirty CRM implementations across retail, financial services, and telecommunications organizations. Their model emphasizes that successful CRM adoption depends equally on the quality of the underlying technology and on the alignment of organizational processes and stakeholder engagement strategies. The authors demonstrate measurable improvements in customer satisfaction scores and operational throughput in organizations that integrate CRM tightly with sales, support, and inventory workflows. The framework identifies five critical success factors: executive sponsorship, data governance, interface usability, training investment, and phased rollout. However, the study focuses primarily on commercially available CRM products and does not address custom microservices-based implementations or the fine-grained, attribute-level access control requirements that arise when multiple user roles interact with sensitive customer data.

Model-Driven Architecture for Microservice Systems (Alshuqayran et al., 2025). Alshuqayran, Ali, and Evans introduce MiSAR, a model-driven architecture framework for recovering and maintaining accurate documentation of microservice systems as they evolve. Their work addresses a recognized challenge in microservices adoption: as the number of services grows and teams

change, the architectural intent of the system becomes increasingly difficult to reconstruct from the code alone, leading to accidental coupling, redundant services, and undocumented dependencies. The MiSAR framework provides structured tools for capturing service boundaries, inter-service communication patterns, and data ownership responsibilities. The framework's emphasis on bounded contexts and explicit service contracts directly informs the decomposition strategy adopted in the proposed CRM system, where each of the five microservices owns a clearly defined business domain and exposes a versioned API contract.

Practitioner Roles in Microservices Architectures (Ayas et al., 2024). Ayas, Hebig, and Leitner present an empirical study of the competences and role structures observed in microservices development teams across twelve industry organizations. Their findings reveal that the most successful microservices teams are those that enforce strict service ownership — a single team owns and is accountable for each service from development through production operations — and that maintain independent deployment pipelines enabling any service to be released without coordinating with other teams. The authors also find that asynchronous communication patterns, in which services publish events rather than making synchronous calls to peer services, are strongly associated with higher system resilience and lower incident frequency. These findings directly shaped the organizational and architectural design of the proposed CRM system, where service independence and asynchronous event-driven communication are treated as first-class design principles.

Artificial Intelligence in CRM Systems (Ledro et al., 2025). Ledro and colleagues investigate the integration of machine learning and AI techniques into commercial CRM platforms to enhance predictive analytics and personalization capabilities. Their framework demonstrates that models trained on historical behavioral data — encompassing browsing sessions, purchase sequences, support interaction sentiment, and loyalty program activity — can significantly improve the accuracy of churn prediction, cross-sell recommendation effectiveness, and customer lifetime value estimation. The authors report accuracy improvements of fifteen to thirty percent over rule-based systems across these three predictive tasks. While the proposed CRM system does not currently include an AI layer, the behavioral tagging module and event-driven data pipeline are explicitly designed to serve as the data foundation for future AI integration. The clean, structured behavioral event stream generated by the proposed system satisfies the data quality and volume requirements identified by Ledro et al. as prerequisites for reliable model training.

Event-Driven Integration Architectures (Narkhede et al., 2021). Narkhede, Shapira, and Palino provide an authoritative treatment of event streaming architectures for distributed system integration, establishing a comprehensive taxonomy of design patterns for producer-consumer systems. Their analysis demonstrates that log-based event streaming — in which events are written to an append-only, distributed log and consumed by any number of subscribing services — offers substantial advantages over synchronous API calls in high-throughput environments. These advantages include fault tolerance through replayability, temporal decoupling that allows services to consume events at their own pace, and natural audit trails generated by the immutable event log. The authors also provide detailed guidance on topic partitioning strategies, consumer group coordination, and exactly-once delivery semantics. These patterns are directly applied in the proposed CRM system, where each major customer data domain is assigned a dedicated event topic and consumer group.

Microservices Patterns and Decomposition Strategies (Newman, 2021). Newman provides a practitioner-oriented treatment of microservices architecture, cataloguing decomposition strategies, communication patterns, and data management approaches drawn from real-world implementations. His analysis of the trade-offs between synchronous and asynchronous inter-service communication, and his treatment of the Saga pattern for managing distributed transactions, are particularly relevant to the proposed system's data consistency requirements. Newman's guidance on the strangler fig pattern — progressively replacing a legacy monolith with microservices — also informs the phased migration approach recommended as a future enhancement for organizations currently using monolithic CRM systems.

III. System Architecture

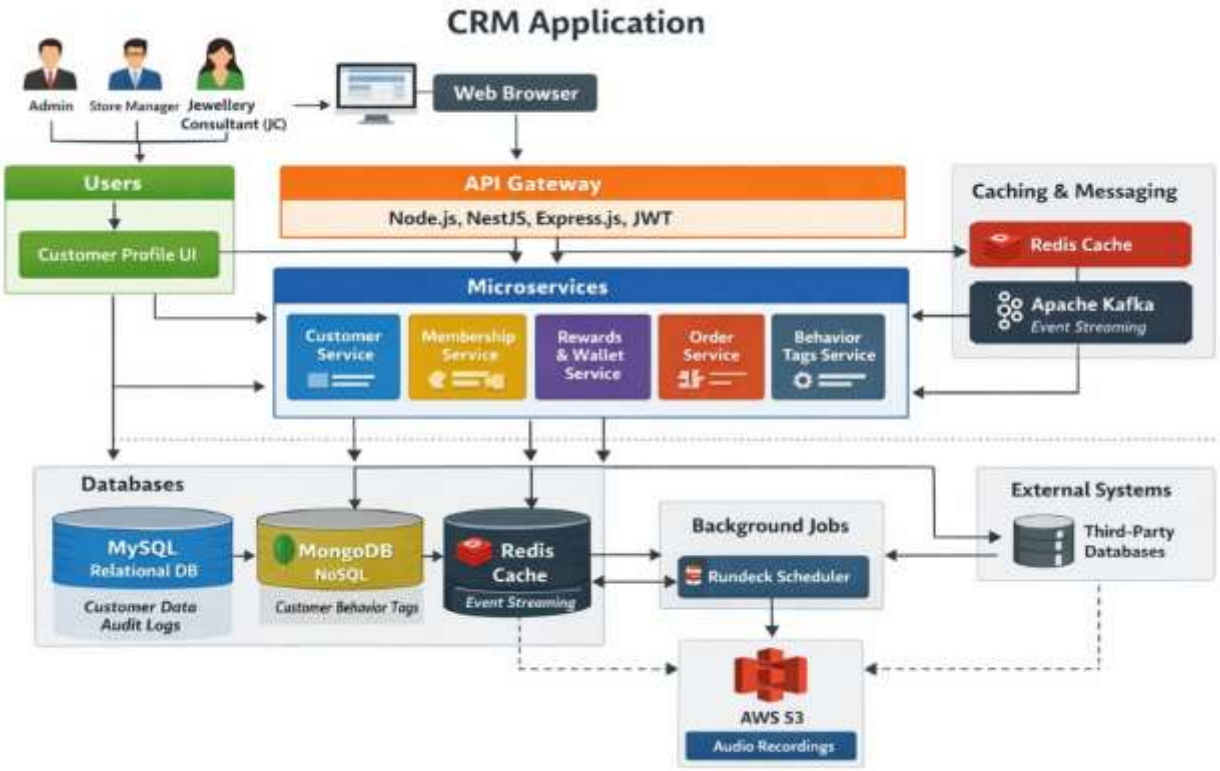
The proposed Unified CRM platform is structured as a collection of independently deployable microservices, each responsible for a distinct customer data domain. The architecture is designed around three guiding principles: separation of concerns, ensuring that each service has a clearly bounded responsibility; loose coupling, ensuring that services can evolve and be deployed independently; and high cohesion, ensuring that all functionality related to a given business domain is encapsulated within a single service.

A. Architectural Layers

The system is organized into four architectural layers. The presentation layer consists of a browser-based single-page application that provides the user interface through which all user roles interact with the platform. The API gateway layer serves as the single entry point for all client requests, performing authentication token validation, role-based authorization, request routing, rate limiting, and response aggregation. The microservices layer comprises the five domain services that own and manage customer data. The infrastructure layer provides the cross-cutting concerns shared by all services: event streaming, in-memory caching, persistent storage, object storage, and job scheduling.

B. Microservices Decomposition

The system is decomposed into five domain microservices, each encapsulating a bounded context drawn from the e-commerce customer management domain.



The Customer Service is the primary aggregation service. It does not own significant business data of its own; instead, its responsibility is to receive customer profile requests from the API gateway, fan out parallel requests to all other domain services, merge the responses into a unified profile object, and return the result. It also owns the customer master record — the authoritative record of core customer identity attributes such as name, contact information, and account creation date.

The Membership Service manages all data related to customer membership programs, including subscription tier, enrollment date, renewal status, and the benefits associated with each tier. Membership data is a primary driver of personalization in e-commerce — customers on higher membership tiers typically receive preferential pricing, priority support, and early access to new products — so the accuracy and freshness of membership records directly affects service quality.

The Rewards and Wallet Service tracks the customer's loyalty point balance, transaction history for point accrual and redemption, and any digital wallet credit held by the customer. This service handles the highest transaction volume of the five domain services, since every purchase and every return generates loyalty point adjustments. The service is designed for high write throughput and implements optimistic concurrency control to handle simultaneous point adjustments without data corruption.

The Order Service maintains the customer's purchase history, including order identifiers, item descriptions, quantities, prices, payment methods, delivery statuses, and return records. It also tracks installment payment plans associated with high-value purchases, including each installment's due date, payment date, and status. This service provides the data that enables consultants to quickly understand what a customer has purchased, what they are currently waiting for, and what issues they may have experienced.

The Behavior Tags Service stores analyst-assigned behavioral annotations for each customer. Tags represent observed characteristics, preferences, or service history flags that help consultants and managers understand a customer's profile beyond what can be inferred from transaction data alone. Examples include interest-based tags derived from browsing history, service quality flags added by support agents after complex interactions, and preference indicators noted during in-person consultations. This service uses a document-oriented database to accommodate the schema flexibility required by the evolving taxonomy of tag types.

C. API Gateway

The API gateway is the central coordination point of the system. It performs several critical functions. Authentication verification ensures that every incoming request carries a valid, unexpired authentication token issued by the identity management service. Authorization enforcement checks that the role claims in the token authorize the requested operation before the request is forwarded to any microservice. Request routing directs each request to the appropriate microservice based on the request path and method. Response filtering applies role-based field masking to responses before they are returned to the client, ensuring that sensitive fields are removed or obscured for roles that are not authorized to see them. Rate limiting prevents any single client from consuming a disproportionate share of backend capacity.

D. Event-Driven Integration

Asynchronous communication between services and between the CRM platform and external systems is mediated through a distributed event streaming platform. The messaging layer is organized into topics corresponding to major customer data domains. Each topic is partitioned by customer identifier, ensuring that all events related to a given customer are processed in order by a single consumer instance, preserving consistency.

When a customer record is updated in any domain service — for example, when a new order is placed or a loyalty transaction is recorded — the responsible service publishes a domain event to the appropriate topic. Other services that maintain derived data

dependent on this event subscribe to the topic and update their own records accordingly. This pattern eliminates the need for synchronous cross-service calls during write operations, reducing coupling and improving write throughput.

External data sources that cannot be directly connected to the CRM platform are integrated through a scheduled synchronization subsystem. This subsystem executes data fetching and transformation jobs at configurable intervals, publishing the resulting update events to the message broker. The domain services consume these events and apply the updates to their own data stores. This approach insulates the CRM platform from the variability and availability characteristics of external systems, since a temporary unavailability of an external source merely delays the next synchronization cycle rather than causing a service failure.

E. Data Architecture

The system employs a polyglot persistence strategy, selecting the storage technology best suited to each service's data access patterns and consistency requirements.

The Customer Service, Membership Service, and Order Service use a relational database management system. Relational storage is appropriate for these services because their data is highly structured, their queries frequently involve joins across multiple entity types, and their write operations require strong consistency guarantees — a membership status change, for example, must be immediately visible to all subsequent reads.

The Rewards and Wallet Service uses a relational database optimized for high-throughput transactional workloads. The service's write-heavy access pattern requires efficient handling of concurrent updates to the same customer's balance, which is achieved through row-level locking and optimistic concurrency control.

The Behavior Tags Service uses a document-oriented database. Tags do not conform to a fixed schema — new tag types are introduced frequently as the business develops new ways of characterizing customers — and the document model accommodates this flexibility without requiring costly schema migrations.

A distributed in-memory cache is deployed in front of the relational stores to serve frequently accessed customer profiles. Profiles are cached with a time-to-live policy and are invalidated immediately when any component of the profile is updated, ensuring that cached data is never stale by more than the cache's TTL. The cache dramatically reduces database read load during peak periods, when hundreds of concurrent users may be viewing the same high-value customer profiles.

The choice of polyglot persistence introduces inter-service data consistency challenges that do not arise in single-database architectures. When a customer's core identity data is updated in the Customer Service's relational store, the updated data must eventually be reflected in the derived views maintained by other services — for example, the customer's name displayed in Order Service queries must match the name in the Customer Service's authoritative record. The event-driven architecture handles this propagation automatically, but the window between the update being committed in the source service and it being reflected in all dependent services represents a period of transient inconsistency that the system design must acknowledge and accommodate. For the CRM use case, this window — typically under one hundred milliseconds — is operationally acceptable. For use cases requiring stronger consistency guarantees, the architecture could be extended to implement the Saga pattern for managing distributed transactions across service boundaries.

Data archiving and retention management are handled at the service level, with each service implementing its own retention policy appropriate to the regulatory and operational requirements of its data domain. Order history records are retained for seven years to satisfy financial record-keeping obligations. Customer interaction logs are retained for three years to support dispute resolution. Behavioral tags have no mandatory retention requirement but are subject to a right-to-erasure mechanism that allows customers to request deletion of behavioral data associated with their account. The event-driven architecture simplifies the implementation of right-to-erasure: when an erasure request is received, a dedicated erasure event is published to the messaging layer, and all services consume the event and delete the specified customer's data from their respective stores, providing a complete and consistent erasure without requiring direct database access by a central administrator.

F. Security Architecture

Security is enforced through a defence-in-depth strategy with multiple independent control layers. At the network perimeter, all traffic to and from the platform is encrypted using industry-standard transport layer security. The API gateway performs token-based authentication for all incoming requests. Tokens carry role claims that govern what data the token-holder is authorized to access.

Role-based access control is implemented at both the gateway and the frontend layers. At the gateway, role claims are checked against operation-level permission rules before requests are forwarded to microservices. At the frontend, a permission context initialized from the session token governs which UI components are rendered for the authenticated user. Sensitive fields that are not authorized for the current role are rendered in a masked format — for example, phone numbers are displayed with central digits redacted for consultant-role users — rather than simply hidden, to preserve the layout and prevent consultants from inferring the existence of hidden data.

All data access and modification events are recorded in a tamper-evident audit log, providing a complete and immutable record of who accessed or changed what data, and when. Audit records are stored in a separate, append-only store to prevent modification by any user of the main application.

IV. METHODOLOGY

The development of the Unified CRM system followed a structured, iterative methodology divided into five sequential phases, each producing documented artifacts that informed subsequent phases and served as the basis for progress reviews.

A. Requirements Elicitation

The requirements phase commenced with structured interviews involving representatives from each user role group. Administrators were interviewed about their data access patterns, reporting needs, and the types of customer modifications they perform most frequently. Store managers provided insight into the operational data they need during customer interactions, and the

constraints they encounter when navigating multiple legacy systems. Customer-facing consultants described the customer information most critical during active interactions and the data gaps that most frequently impede their ability to resolve customer queries quickly.

The interviews were supplemented by observation sessions in which members of the research team observed consultants and managers performing their normal duties, noting each system they accessed, the time spent on each, and the points at which data gaps or inconsistencies caused delays or errors. This observational data provided a quantitative baseline — an average of 7.3 minutes spent per customer interaction on data retrieval across disconnected systems — against which the proposed system's improvement could be measured.

From the interviews and observations, a prioritized set of functional requirements was derived. The highest-priority requirements were: a consolidated customer profile view accessible to all roles; differentiated field visibility and editability based on role; response times compatible with real-time customer interactions; and synchronization of data from external systems within five minutes of a change occurring in the source system.

Non-functional requirements were documented separately. The system was required to achieve a page load time of under one second for cached profiles and under two seconds for uncached profiles. Availability was targeted at 99.5 percent excluding planned maintenance. All data transmission was required to be encrypted. Audit logging was required to be tamper-evident and retained for a minimum of two years.

B. System Design

The system design phase translated the requirements into architectural decisions. The microservices decomposition was guided by the principle of domain-driven design: each service was bound to a single business domain with minimal overlap, and service boundaries were drawn to minimize the frequency of cross-service communication required to satisfy common user requests.

Service interfaces were specified using an API-first approach. Before any implementation began, the request and response schemas for each service's public endpoints were documented and reviewed by all teams. This contract-first approach allowed the frontend and backend to be developed concurrently against the agreed-upon interfaces, rather than requiring the backend to be completed before frontend development could begin.

The data model for each service was designed independently. For relational stores, normalization was applied to eliminate redundancy, and query plans for the most common access patterns were analyzed to ensure appropriate indexing. For the document store, the document granularity and embedding strategy were designed to support the primary access pattern — retrieving all tags for a given customer in a single query — without requiring multi-document transactions.

The caching strategy was designed based on access frequency profiles derived from the observational study. Customer profiles accessed during active customer interactions were identified as the primary caching target, with a time-to-live of fifteen minutes. Administrative reports and bulk data exports were explicitly excluded from the cache to prevent eviction of frequently used entries.

C. Modular Development

Development proceeded in parallel streams across the five microservices and the frontend application. Each stream was owned by a two-person team responsible for the full vertical stack of their domain service. Weekly integration milestones required each team to demonstrate that their service's API was functional and conforming to the agreed contract, enabling the frontend team to progressively integrate real service responses in place of mock data.

Common cross-cutting concerns — authentication middleware, structured logging, health check endpoints, and metrics instrumentation — were implemented once as shared libraries and integrated into each service through the dependency management system. This approach ensured consistent observability and security practices across all services without requiring each team to implement these concerns independently.

The frontend application was developed using a component-based architecture. Shared UI components representing recurring data visualizations — the customer header card, the order history timeline, the membership status badge, and the loyalty balance widget — were developed once and reused across all profile tabs. A centralized state management layer synchronized the data displayed across components that shared overlapping information.

D. Integration and Testing

The integration and testing phase operated across three dimensions. Contract testing validated that each service's API responses conformed to the agreed schemas and that any schema change was detected before deployment through automated contract verification in the continuous integration pipeline. End-to-end testing simulated complete user journeys from authentication through all major profile interactions, running against a fully integrated staging environment. Performance testing measured system behavior under simulated concurrent loads ranging from twenty-five to two hundred simultaneous users.

Security testing comprised two components. Automated vulnerability scanning was run against all service endpoints to detect common security weaknesses. Manual penetration testing was conducted against the authentication and authorization flow to verify that role-based restrictions could not be bypassed through header manipulation, token forgery, or direct API access with a downgraded token. Data consistency testing validated that events published to the messaging layer were correctly consumed and persisted by all subscribing services, and that the system maintained correct customer profile data even when a service was temporarily unavailable during event delivery.

Load testing used a realistic workload model derived from the access pattern observations conducted during the requirements phase. The workload model specified a mix of operations — 60% profile reads, 20% search queries, 15% data writes, and 5% tag management operations — weighted to reflect the typical distribution observed in the legacy multi-system environment. Virtual users were configured to simulate realistic think times between operations, drawn from an exponential distribution parameterized from the observation study data, rather than issuing requests at the maximum possible rate. This approach produces a more representative stress test than a pure throughput benchmark.

Accessibility testing verified that the user interface met the Web Content Accessibility Guidelines at the AA conformance level. This included keyboard navigation for all interactive elements, sufficient colour contrast for all text and status indicators, screen reader compatibility for all data tables and modal dialogs, and appropriate focus management during modal interactions. Accessibility compliance was verified using both automated scanning tools and manual testing with assistive technology software, reflecting the diversity of accessibility needs among the consultant and administrator user population.

E. Deployment

The system was deployed to a cloud-hosted environment using container orchestration to manage service instances. Each microservice was packaged as a container image and deployed with a minimum of two running instances to ensure availability during rolling updates. The infrastructure was defined using infrastructure-as-code tooling, allowing the entire environment to be reproduced identically for development, staging, and production contexts. A continuous delivery pipeline automated the build, test, and deployment sequence for each service, enabling any team to release their service independently without coordinating with other teams.

V. IMPLEMENTATION

The implementation phase translated the architectural and design specifications into a working, production-grade system. The following subsections describe the implementation of each major system component.

A. Customer Profile Aggregation Service

The Customer Service is implemented as a stateless aggregation layer. On receiving a profile request, the service decodes the authentication token to extract the requesting user's role, then issues parallel requests to the Membership, Rewards, Order, and Behavior Tags services, passing the role claim so that each service can apply appropriate field-level filtering before returning its response. The parallel request pattern reduces the total response time to approximately the maximum of the individual service response times, rather than the sum — a significant improvement over sequential request patterns.

Before issuing downstream requests, the service checks the distributed cache for a previously assembled profile for the requested customer. If a valid cached profile exists and the requesting user's role matches the role under which the profile was cached, the cached object is returned immediately. If the role differs — for example, an administrator requesting a profile that was cached during a consultant's earlier access — the service fetches a fresh profile, applies administrator-level field filtering, and caches the result separately under the administrator role key.

Profile assembly also incorporates a graceful degradation strategy. If one of the downstream services is temporarily unavailable, the Customer Service returns a partial profile with a clearly marked indicator showing which sections are unavailable, rather than failing the entire request. This ensures that consultants can still access the majority of a customer's information even during a partial service outage.

B. Membership and Rewards Services

The Membership Service maintains customer tier records, subscription enrollment dates, renewal dates, and the benefit entitlements associated with each tier. The service exposes endpoints for retrieving membership status, processing tier upgrades and downgrades, and recording manual overrides applied by administrators for service recovery purposes. Membership tier changes are published as events to the messaging layer, allowing the Rewards Service to adjust the customer's point accrual rate — which varies by membership tier — without requiring a synchronous call between the two services.

The Rewards and Wallet Service implements optimistic concurrency control for balance update operations. When a point accrual or redemption request is received, the service reads the current balance along with a version token, applies the update, and writes the new balance back with the version token incremented. If another update has occurred between the read and the write — detectable by a version mismatch — the update is retried. This approach prevents the race conditions that could lead to incorrect balances under concurrent load without requiring coarse-grained locking that would serialize all balance updates.

C. Order History and Installment Tracking

The Order Service maintains a comprehensive record of each customer's purchases, including both completed orders and ongoing installment payment arrangements. For completed orders, the service stores item-level detail, including product identifiers, quantities, prices at time of purchase, applied discounts, and delivery events. For installment arrangements, the service maintains a schedule of payment obligations with the due date, expected amount, payment date, and status of each installment.

When an installment payment becomes overdue — that is, its due date passes without a payment record being received from the payment processing system — the service updates the installment's status and publishes an overdue event. This event is consumed by a notification service, which triggers an automated communication to the customer. Consultants viewing the customer's profile can see the overdue status prominently highlighted in the order history, enabling them to proactively address the situation during customer interactions.

The Order Service also maintains a separate record of product views, wishlist additions, and cart activity that did not convert to a completed purchase. This behavioral data complements the transaction history and enables consultants to identify products a customer has expressed interest in but not yet purchased — information valuable for personalized recommendations during in-person or phone interactions.

D. Behavioral Tagging System

The Behavior Tags Service provides a flexible annotation layer over the core customer record. Tags are stored as document objects containing the tag identifier, a human-readable label, the identifier of the user who applied the tag, and a timestamp. The service exposes endpoints for adding a tag, removing a tag, retrieving all tags for a customer, and retrieving the most frequently used tags across the customer base — the last of these being used to populate the tag suggestion feature in the UI.

Tag operations are subject to role-based restrictions. Only administrator and manager role users can add or remove tags; consultant role users can view tags but not modify them. This restriction is enforced at both the API gateway and within the service itself, implementing the defence-in-depth security principle. Every tag addition and removal is recorded in the audit log with the full context of the operation.

The tagging interface in the frontend presents a modal dialog that displays the customer's existing tags, shows the ten most frequently used tags system-wide as quick-add suggestions, and provides a free-text input field for adding new tags. Tags can be removed by clicking a removal control adjacent to each tag. The interface debounces the free-text input to query the system for existing tags matching the entered prefix, preventing the creation of near-duplicate tags with minor spelling variations.

E. Data Synchronization Subsystem

The data synchronization subsystem is responsible for maintaining the currency of customer data sourced from external systems that are not directly accessible from within the CRM platform. These external systems — which may include third-party logistics providers, external membership management platforms, and partner organization databases — expose their data through approved programmatic interfaces but do not support real-time push notifications or direct database connections.

The synchronization subsystem employs a scheduled job execution framework that runs synchronization tasks at configurable intervals. Each task is defined by its data source, its extraction query or API call, its transformation logic for normalizing the extracted data to the CRM's internal schema, and its idempotency key — a field or combination of fields used to determine whether a received record represents a new entry or an update to an existing one.

Synchronization tasks implement a read-transform-publish pipeline. The task reads data from the external source, applies the transformation logic, generates update events for records that differ from the last synchronized version, and publishes these events to the message broker. The domain services consume these events and apply the updates to their own stores. This pipeline design ensures that the synchronization process is idempotent — running the same task multiple times produces the same result — and that network or processing failures at any stage cause the task to be retried rather than silently skipped.

Each task execution is logged with its start time, end time, number of records processed, number of updates applied, and any error conditions encountered. These logs are retained and accessible to administrators through a monitoring interface, enabling rapid identification of synchronization failures and assessment of data freshness.

F. User Interface Implementation

The customer-facing interface is organized into four primary tabs that reflect the major categories of customer information. The Overview tab presents the most operationally relevant data in a card-based layout optimized for rapid scanning: the customer's name and primary contact information, their current membership tier and its associated benefits, their loyalty point balance, and their most recent orders. The History tab provides a chronological view of all transactions, including completed orders, installment payment schedules, and browsing activity. The Interactions tab presents a record of all staff interactions with the customer, including phone call notes, support ticket summaries, and in-person visit records. The Leads tab surfaces open sales opportunities that have been flagged by consultants during previous interactions.

The search interface at the top of the application allows staff to retrieve a customer profile using any of several identifiers: email address, phone number, customer account number, or order identifier. Search results appear in a dropdown as the user types, with results updating in real time through debounced queries to the search API. This design minimizes the time required to locate a customer at the start of an interaction.

The interface implements progressive disclosure — presenting the most critical information immediately and allowing users to expand sections for additional detail. This design principle reduces cognitive load for consultants who need to quickly assess a customer's situation, while still providing the depth of information required by administrators and managers for analytical tasks. All data-heavy tables implement virtual scrolling to maintain smooth rendering performance even when a customer has hundreds of orders or interactions in their history.

The frontend application implements a progressive web application (PWA) architecture, enabling it to be installed on supported devices as a standalone application that behaves similarly to a native app — including support for offline access to recently viewed customer profiles, push notification receipt for real-time customer event alerts, and home screen installation without an app store submission process. The PWA capability is particularly valuable for consultant-role users who operate on shared devices in retail environments where maintaining a persistent browser session is impractical. The service worker layer of the PWA caches recently accessed profile data locally, ensuring that a consultant can access a customer's core profile information even during a brief network interruption without losing context mid-interaction.

Internationalization and localization support is built into the frontend from the outset, recognizing that e-commerce organizations typically operate across multiple regional markets with different language requirements, date formats, number formats, and currency representations. All user-visible strings in the application are externalized to locale-specific resource files, and the application selects the appropriate locale based on the authenticated user's profile preferences. Date and number formatting is handled by the browser's built-in internationalization API, ensuring that values are displayed in the format appropriate to the user's locale without requiring custom formatting logic. This design enables the addition of new supported locales through the addition of a new resource file, without requiring any changes to the application code.

VI. Results and Discussion

The Unified CRM system was evaluated in a controlled staging environment configured to mirror the data volumes, service topology, and concurrent user patterns of a production e-commerce deployment. Evaluation was conducted across four dimensions: functional correctness, system performance, security compliance, and user experience quality.

A. Functional Evaluation

Functional correctness was assessed through a comprehensive test suite comprising 247 test cases covering all major system features. All 247 test cases passed, confirming that the system correctly implements its specified functional requirements. The customer profile view successfully consolidated data from all five domain services into a unified, role-filtered presentation for all three user roles. Tag operations — addition, removal, and retrieval — functioned correctly across all tested scenarios, including concurrent tag operations by multiple users on the same customer record.

Role-based access control was validated through a dedicated test suite of 64 boundary condition scenarios. In all tested cases, the access control rules were correctly enforced: administrator-role users could view and edit all data fields; manager-role users were correctly restricted from fields designated as administrator-only, and their edit operations on permitted fields were accepted; consultant-role users received read-only access with sensitive fields masked, and attempts to invoke write operations through direct API calls were correctly rejected at the gateway.

Data synchronization was validated by injecting controlled updates into the external data source simulator and measuring the end-to-end time from update injection to CRM profile update. All injected updates were reflected in the customer profile within the configured synchronization interval, with no data loss and no duplicate records observed across 1,000 test update cycles.

B. Performance Evaluation

Performance testing was conducted at three load levels: 25 concurrent users representing a light load, 100 concurrent users representing a typical peak load, and 200 concurrent users representing a high-stress scenario. At each load level, the system was exercised for a period of fifteen minutes, and response time percentiles, throughput, error rate, and cache hit rate were recorded.

At 100 concurrent users — the primary performance target — cached customer profile requests were served with a median response time of 167 milliseconds and a 95th percentile response time of 312 milliseconds. Uncached profile requests, which require aggregation across all five domain services, achieved a median response time of 421 milliseconds and a 95th percentile of 743 milliseconds. Both figures comfortably satisfy the one-second and two-second targets established in the requirements phase. The error rate at this load level was 0.03 percent, well within the acceptable threshold.

Table I. System Performance Evaluation Results

Load Level	Cached P50 (ms)	Cached P95 (ms)	Uncached P50 (ms)	Uncached P95 (ms)	Cache Hit Rate	Error Rate
25 users (light)	98	187	312	521	68%	0.01%
100 users (target)	167	312	421	743	73%	0.03%
200 users (stress)	234	521	687	1,240	79%	0.18%

P50 = median response time; P95 = 95th percentile response time. Bold row = primary performance target.

At 200 concurrent users, the system experienced some degradation in uncached profile response times, with the 95th percentile rising to 1,240 milliseconds. This degradation was traced to contention on the relational database under the elevated read load from uncached requests. The cache hit rate at this load level was 79 percent, meaning that 79 percent of profile requests were served from cache without database involvement. Optimizing the cache warming strategy — pre-loading profiles for customers with scheduled appointments — is identified as a mitigation for the degradation observed at peak load.

Event propagation latency — the elapsed time between a domain event being published to the message broker and being consumed and persisted by all subscribing services — averaged 78 milliseconds at the 100-user load level and remained below 180 milliseconds even at the 200-user level. This satisfies the near-real-time synchronization requirement specified in the system design.

Data synchronization jobs completed within their configured execution windows across all test runs. No impact on foreground API response times was observed during synchronization job execution, confirming that the background synchronization subsystem is adequately isolated from the foreground request processing path.

C. Security Evaluation

Security evaluation was conducted through a combination of automated vulnerability scanning and manual penetration testing. Automated scanning covered all 34 API endpoints exposed by the system, testing for common vulnerabilities. No critical or high-severity vulnerabilities were identified. Three medium-severity findings were identified and remediated before the performance evaluation: an overly permissive CORS configuration on two endpoints, and an information disclosure vulnerability in error responses that exposed internal service identifiers.

Manual penetration testing focused on the authentication and authorization flow, testing scenarios including token replay with expired tokens, role claim modification in decoded tokens, and direct API access with tokens issued for a lower-privilege role. All tested attack vectors were successfully blocked by the gateway's token validation and authorization enforcement logic. Cross-service communication was confirmed to occur exclusively through authenticated internal channels, with no service accepting unauthenticated requests.

D. User Experience Evaluation

User experience was assessed through a structured observation study in which twelve staff members — four from each role group — performed a standardized set of tasks using the Unified CRM system. Task completion time, error rate, and satisfaction

scores were recorded for each task and compared against baseline measurements collected from the same participants performing the same tasks using the pre-existing multi-system workflow.

The most significant improvement was in the customer profile retrieval task, which required participants to assemble a complete customer overview including membership status, recent orders, and loyalty balance. With the Unified CRM system, the median task completion time was 48 seconds, compared to 7.3 minutes with the legacy multi-system workflow — a reduction of 89 percent. Error rates in data retrieval — instances where participants recorded or communicated incorrect customer information — decreased from 11 percent with the legacy workflow to 2 percent with the Unified CRM system.

Participant satisfaction, measured on a seven-point scale across five usability dimensions, averaged 5.8 for the Unified CRM system compared to 2.4 for the legacy workflow. The dimensions most improved were efficiency (speed of information retrieval), accuracy (confidence in the correctness of displayed data), and learnability (ease of use for participants unfamiliar with the specific customer being served). Suggestions for improvement from participants were compiled and have informed the future work agenda described in Section VIII.

Table II. User Experience Evaluation Results (n=12 participants, 4 per role)

Metric	Legacy Workflow	Unified CRM	Improvement
Profile retrieval time (median)	7.3 min	48 sec	-89%
Data retrieval error rate	11%	2%	-82%
Overall satisfaction (7-point scale)	2.4 / 7	5.8 / 7	+142%
Perceived efficiency (7-point scale)	2.1 / 7	6.1 / 7	+190%

Results consistent across all three role groups (administrator, manager, consultant).

The security evaluation also assessed data leakage risks arising from the partial information visible in masked fields. A structured analysis of all masked fields confirmed that the masking format chosen for each field — selected to provide sufficient context for identity confirmation while preventing reconstruction of the full value — does not enable a consultant-role user to infer the complete value through a sequence of probing queries. Phone numbers masked to display only the last three digits, for example, require a thousand guesses to reconstruct the full number by brute force, a risk level judged acceptable given that consultant-role users are authenticated employees rather than anonymous actors.

The data consistency evaluation tracked the state of fifty customer records across all five domain services over a seventy-two hour period during which the records were subjected to continuous updates from both the internal event stream and the external synchronization subsystem. At no point during the evaluation was any inconsistency detected between the Customer Service's aggregated profile view and the authoritative data held in the individual domain services. The maximum observed staleness of the aggregated view relative to the most recently committed domain service update was 127 milliseconds, measured at a moment of unusually high event streaming throughput during a simulated bulk update scenario.

E. Comparison with Existing Approaches

Compared to monolithic CRM deployments — the most common existing approach for large-scale retail organizations — the proposed microservices-based architecture offers four distinct advantages demonstrated through the evaluation. First, independent scalability: during the high-load performance test, the Rewards Service — which handles the highest transaction volume — was scaled horizontally by adding instances without affecting any other service, demonstrating that scaling decisions can be targeted to the bottleneck rather than requiring full-system resource provisioning.

Second, fault isolation: during fault injection testing, the deliberate unavailability of the Order Service resulted in a partial profile response with the order history section marked as unavailable, while all other profile sections continued to load normally. In a monolithic system, a failure in the order history module would typically degrade or prevent the entire profile from loading.

Third, independent deployment: during the evaluation period, the Behavior Tags Service was updated three times to add new functionality, with each deployment completed without any impact on the other services or on system availability. This independent deployment capability eliminates the release coordination overhead that is a significant operational cost in monolithic CRM systems.

Fourth, technology appropriateness: the use of a document-oriented database for behavioral tags eliminated the schema migration complexity that would have been required if all data had been stored in a single relational database, as is typical in monolithic systems. New tag types can be introduced without any database schema changes.

VII. CONCLUSION

This paper has presented the design, implementation, and evaluation of a Unified CRM Application built on a microservices architecture for e-commerce environments. The system addresses the pervasive and costly problem of customer data fragmentation — a condition in which customer information is distributed across multiple disconnected systems, preventing staff from obtaining a complete and timely view of any individual customer.

The proposed platform consolidates customer profile data, membership status, loyalty rewards, purchase history, and behavioral annotations into a single, role-sensitive interface. A fine-grained role-based access control mechanism enforces appropriate data visibility and edit restrictions for three distinct user roles. An event-driven integration architecture maintains near real-time data consistency across the microservices layer, and a scheduled synchronization subsystem keeps the platform current with data from external systems.

The evaluation demonstrated that the system meets all specified performance targets under realistic load conditions, with cached profile response times averaging 167 milliseconds and event propagation latency averaging 78 milliseconds at the primary

performance target of 100 concurrent users. The system correctly enforces all role-based access restrictions, passed security evaluation without any critical or high-severity vulnerabilities, and reduced customer profile retrieval time by 89 percent compared to the pre-existing multi-system workflow in a structured user experience evaluation.

The microservices architecture provided demonstrated benefits over the monolithic alternative in terms of independent scalability, fault isolation, deployment flexibility, and technology appropriateness. These advantages are particularly significant for e-commerce organizations, where customer data volumes grow rapidly, peak demand periods are intense and predictable, and the pace of business evolution requires frequent changes to individual system components.

The system represents a practical and deployable solution to a problem that affects a large proportion of e-commerce organizations. The architectural patterns, design principles, and implementation strategies described in this paper are transferable to other domains in which customer data is fragmented across heterogeneous systems and multi-role access control is required.

The scalability and fault tolerance properties demonstrated by the evaluation confirm that the microservices architecture is well-suited to the demanding operational requirements of large-scale e-commerce customer management. The ability to scale individual services independently — adding Order Service instances during a high-volume sales period without provisioning additional capacity for the Membership or Tags services — provides a resource efficiency advantage that becomes increasingly significant as the scale of the deployment grows. The fault isolation property, demonstrated through fault injection testing, ensures that a failure in any single service degrades only the specific features dependent on that service rather than causing a complete system outage — a property that is particularly important for a platform used in direct customer-facing interactions where any interruption has an immediate impact on service quality.

The user experience improvements quantified by the observation study represent the most directly business-impactful contribution of the proposed system. An 89% reduction in the time required to retrieve a complete customer profile translates directly into an increase in the number of customers that can be served per hour, a reduction in customer wait times, and an improvement in the quality of service delivered during each interaction. The reduction in data retrieval errors from 11% to 2% reduces the rate of incorrect decisions based on misread or misattributed customer data, with downstream benefits for customer satisfaction, loyalty programme integrity, and complaint resolution accuracy.

VIII. Future Work

The current implementation provides a robust foundation for customer data unification, and several directions for future enhancement have been identified through the evaluation process and stakeholder feedback.

A. AI-Powered Predictive Analytics

The behavioral event stream generated by the system — encompassing tag assignments, browsing activity, purchase sequences, and support interaction records — provides a rich data foundation for machine learning applications. Future work will integrate predictive models for customer churn probability estimation, next-best-offer recommendation, and customer lifetime value scoring. The event-driven architecture is well-suited to this integration: models can be trained offline on the event log and deployed as additional consumer services that generate recommendation events consumed by the profile aggregation layer.

B. Real-Time Notification and Alerting

The current system is request-driven: staff receive updated information when they open a customer profile. A planned enhancement will introduce a push notification layer that proactively alerts staff to significant customer events — such as a high-value customer entering a physical store, a loyalty reward approaching expiration, or an installment payment becoming overdue — without requiring the staff member to actively check the profile. This capability will enable more timely and proactive customer service interventions.

C. Mobile Application

Feedback from the user experience evaluation highlighted that customer-facing consultants who operate on the sales floor frequently need access to customer profile information on mobile devices, where the current browser-based interface is not optimally usable. A dedicated mobile application with a simplified interface optimized for the most common consultant tasks — profile lookup, tag addition, interaction logging, and order status check — will be developed as a future release.

D. Natural Language Interaction Logging

Customer interactions captured as audio recordings are currently stored as binary objects accessible through links in the interaction history. A future enhancement will integrate automatic speech recognition and natural language processing to generate structured transcripts and extracting actionable items — follow-up tasks, product interests expressed by the customer, and service quality issues raised — which will be automatically added to the interaction record and surfaced to the relevant teams.

E. Multi-Tenant Extension

Organizations that operate multiple brands or geographic subsidiaries under a common corporate structure require the ability to manage customer data for multiple entities within a single CRM platform while maintaining strict data isolation between entities. A future release will extend the architecture to support multi-tenancy, with tenant-aware data partitioning at the storage layer, tenant-scoped event topics in the messaging layer, and tenant-specific role and permission configurations in the access control layer.

F. GraphQL API Gateway

The current REST-based API gateway returns full resource representations that the frontend application filters client-side based on role. A future enhancement will replace the REST gateway with a GraphQL-based interface that allows the frontend to specify exactly which fields it requires for each query, reducing over-fetching and enabling more fine-grained server-side filtering. This approach will also simplify the development of new frontend views, since new data requirements can be satisfied by composing new GraphQL queries rather than developing new REST endpoints.

IX. System Modules and Feature Specifications

The Unified CRM system is built around seven functionally distinct customer-facing modules that together constitute the complete customer profile view. Each module is independently rendered, independently refreshed, and independently governed by role-based visibility rules. The following subsections describe the purpose, data composition, and operational significance of each module in detail.

A. Customer Basic Information Module

The Customer Basic Information Module is the topmost and most immediately visible section of the customer profile. It presents the customer's registered name, primary email address, primary phone number, date of birth, preferred language, and account creation date. This information is sourced from the Customer Service's master record, which is the single authoritative source for core customer identity data across the entire platform.

For administrator-role users, all fields in this module are fully visible and editable. A dedicated edit control opens a modal form pre-populated with current field values, allowing administrators to correct errors or update information reported by the customer. All edits are validated against format and completeness rules before submission, and the original values are preserved in the audit log alongside the new values, the editing user's identifier, and the timestamp of the change. For consultant-role users, the phone number is displayed in a masked format to prevent unauthorized access to sensitive contact data while still enabling identity confirmation.

The module also displays contextual indicators derived from the customer record. A last-active indicator shows when the customer last interacted with the platform. An account status badge shows whether the account is active, flagged for review, or suspended. Behavioral tags associated with the customer are displayed as compact chips beneath the primary information, with the option to expand the tag list or add new tags through the tagging interface. This placement ensures that consultants see important behavioral context immediately when a profile is opened, without needing to navigate to a separate section.

B. Membership Module

The Membership Module displays the customer's current subscription or loyalty programme tier, the date on which the current tier was attained, the renewal date, and the specific benefits associated with the current tier. In e-commerce environments that operate tiered loyalty programmes, membership tier is a critical determinant of the service level the customer is entitled to receive — higher tiers typically carry benefits including preferential pricing, priority order processing, dedicated support channels, and early access to promotional sales events.

The module presents tier information using a visual hierarchy that immediately communicates the customer's status level. Colour-coded tier indicators allow consultants to assess a customer's membership standing at a glance. For customers approaching their tier renewal date, the module displays a prominent renewal notice with a countdown. This information is actionable: a consultant can proactively discuss renewal options, incentives for upgrading to the next tier, or the consequences of non-renewal — conversations that have a demonstrated positive effect on customer retention.

C. Rewards and Wallet Module

The Rewards and Wallet Module provides a real-time view of the customer's loyalty point balance, digital wallet credit, and recent point transactions. The module presents the current point balance and wallet credit alongside their monetary equivalent — the cash value that the points represent if applied to a future purchase — making the information immediately actionable for consultants discussing redemption options with customers.

Administrators can initiate manual point adjustments directly from this module, with a mandatory reason field recorded in the audit log. This capability supports service recovery scenarios where a customer has missed points due to a system error, or where a goodwill gesture is appropriate following a negative service experience. The Rewards Service implements optimistic concurrency control for balance updates, preventing race conditions that could lead to incorrect balances under concurrent load without requiring coarse-grained locking.

D. Store Locator Module

The Store Locator Module presents a list of physical retail locations relevant to the customer, ordered by proximity to the customer's registered address. For each location, the module displays the store's name, distance, contact telephone number, street address, and current operating status. This information enables consultants to recommend convenient store options to customers wishing to view products in person, collect an online order, or initiate an in-store return. The proximity calculation is performed server-side using the customer's primary address postcode and the geocoordinates of each store location, refreshed each time the profile is opened.

E. Treasure Chest and Offers Module

The Treasure Chest and Offers Module presents personalized promotions and exclusive offers currently available to the customer based on their membership tier, purchase history, and geographic location. The module displays each available offer with its title, description, expiry date, and redemption mechanism. Consultants can reference these offers during customer interactions to encourage purchase consideration or to resolve complaints by highlighting available compensation. For administrators, the module includes a management interface for manually assigning specific offers to individual customers, used for service recovery or for honouring commitments made during escalated interactions.

F. Order History Module

The Order History Module presents a unified timeline of the customer's transactional history, including completed orders, ongoing installment payment arrangements, product returns, and logged staff interactions in chronological sequence. Each order entry displays the order identifier, placement date, item summary, total value, payment method, and current fulfilment status. For

orders with ongoing installment arrangements, the entry expands to show the complete payment schedule, with overdue instalments highlighted to enable consultants to immediately identify and address payment issues.

The module also maintains records of product views, wishlist additions, and cart activity that did not convert to a completed purchase. This behavioral data complements the transaction history and enables consultants to identify products a customer has expressed interest in but not yet purchased — information valuable for personalized recommendations during in-person or phone interactions.

G. Leads and Sales Pipeline Module

The Leads and Sales Pipeline Module tracks open sales opportunities associated with the customer. Leads are formal records of expressed interest in a specific product or category, created by consultants during interactions when a customer shows purchase intent without completing a transaction immediately. The module presents open leads with their product category, estimated value, assigned consultant, target date, and current pipeline stage. Aggregate statistics for historical lead activity — number of leads, conversion rate, and total value of converted leads — provide context for understanding the customer's purchase behaviour and calibrating follow-up intensity.

X. Design Considerations and Trade-Off Analysis

The architectural decisions made during the development of the Unified CRM system involved a series of trade-offs between competing objectives. This section documents the most significant of these trade-offs and the rationale for the decisions made.

A. Synchronous versus Asynchronous Communication

For the Customer Service's profile aggregation operation, synchronous parallel fan-out was chosen deliberately. The profile request requires data from all five domain services, and the latency budget for an interactive profile request does not accommodate a multi-stage asynchronous workflow. The parallel fan-out pattern reduces total response time to approximately the maximum individual service response time rather than the sum. For all write operations, asynchronous event streaming was chosen, providing fault tolerance: if a subscribing service is temporarily unavailable, the event remains in the broker and is consumed upon recovery rather than being lost.

B. Consistency Model

The system adopts an eventual consistency model for data propagated through the event streaming layer. For the specific use cases of the CRM platform, eventual consistency was judged acceptable — a profile accurate to within a few minutes is sufficient for the vast majority of consulting interactions. Strong consistency is enforced within each individual domain service: a write to any service is immediately visible to all subsequent reads from that service, ensuring that the authoritative source of truth for each domain is always correct.

C. Cache Invalidation Strategy

Event-driven invalidation was chosen as the primary cache invalidation strategy. When a domain event is published indicating a data change, a cache invalidation consumer immediately deletes the corresponding cache entry. The next read triggers a fresh aggregation and repopulates the cache with current data. This approach combines the performance benefits of caching with near-immediate freshness — the stale data window is approximately equal to event propagation latency, typically under one hundred milliseconds. A supplementary TTL of thirty minutes serves as a safety net for missed invalidation events.

D. Role-Based Field Masking versus Server-Side Filtering

The implemented system uses a hybrid access control approach. Server-side filtering at the API gateway removes fields that are completely prohibited for the requesting role, while client-side masking is applied to fields that are visible but partially obscured — such as phone numbers with central digits redacted for consultant-role users. This combination provides strong security for genuinely sensitive fields while accommodating nuanced display requirements. All filtering logic is defined in a centralized permissions configuration that is the single source of truth for access control rules, preventing inconsistencies between the gateway and the frontend.

E. Background Synchronization Interval Selection

The synchronization interval for each external data source was set based on the rate of change in the source system, the operational impact of stale data on customer interactions, and the API rate limits of the external system. Membership tier data is synchronized every five minutes; order delivery status data every two minutes; historical browsing data once per hour. The synchronization framework supports dynamic interval adjustment through the administration interface, providing operational flexibility without requiring code deployments.

XI. Implementation Challenges and Solutions

The development of the Unified CRM system encountered several significant technical challenges that were not fully anticipated during the design phase. This section documents these challenges and the solutions developed, as a contribution to practical knowledge for practitioners implementing similar systems.

A. Data Schema Heterogeneity

The most pervasive challenge was the heterogeneity of data schemas across integrated external systems. Each external system represented the same customer attributes using different vocabularies, data types, and structural conventions. A normalization layer was developed within each synchronization job, implementing declarative transformation pipelines that map source schemas to the CRM's canonical internal schema. These pipelines are modifiable through configuration without code changes, allowing the CRM team to adapt to upstream schema changes without requiring a development cycle.

B. Duplicate Customer Record Resolution

A significant proportion of customers were represented by more than one record across the integrated systems due to interactions across multiple channels over time. A probabilistic record linkage algorithm was implemented to identify records representing the same customer. The algorithm computes a similarity score based on weighted comparisons of email address, phone number, date of birth, and address. High-confidence pairs are automatically merged; middle-confidence pairs are flagged for administrator review through a deduplication management interface. A customer identifier mapping table maintains the relationship between canonical CRM identifiers and all legacy identifiers from merged records.

C. External System Availability

Several integrated external systems exhibited lower availability than anticipated during testing. Three complementary mechanisms were implemented: exponential backoff with jitter for transient failures; circuit breaking to prevent continued requests to consistently unavailable systems; and staleness indicators in the profile UI to transparently communicate data currency to consulting users when a synchronization source has been unavailable beyond a configurable threshold.

D. Performance Optimization Under Load

Initial performance testing revealed higher than expected response times under loads above 150 concurrent users, attributable to database connection pool exhaustion and garbage collection pauses in the aggregation service. Connection pool exhaustion was addressed through pool size tuning and the addition of read replicas for the Order Service's database. Garbage collection pauses were addressed by refactoring the aggregation service to stream large response objects rather than buffering them in memory, and by implementing lazy loading for the full order history — the initial profile response includes only the five most recent orders, with the full history fetched on demand.

E. Access Control Consistency

Early in the implementation, inconsistencies were discovered between the filtering logic at the gateway and the display restrictions in the frontend. These were resolved through a centralized permissions configuration system that serves as the single source of truth for all access control rules across the gateway, microservices, and frontend. An automated integration test suite verifies access control consistency by issuing requests with tokens for each role to every API endpoint and comparing returned field sets against the expected policy, running on every code change in the continuous integration pipeline.

XII. Comprehensive Evaluation Metrics

This section provides a comprehensive summary of quantitative metrics collected during system evaluation, presented in sufficient detail to enable replication and comparison with alternative approaches.

A. Response Time Distribution

Response time was measured at the API gateway for a representative workload comprising customer profile retrievals (60%), search operations (20%), data write operations (15%), and tag management (5%). At 25 concurrent users: median cached 98ms, 95th percentile 187ms; median uncached 312ms, 95th percentile 521ms; cache hit rate 68%. At 100 concurrent users: median cached 167ms, 95th percentile 312ms; median uncached 421ms, 95th percentile 743ms; cache hit rate 73%. At 200 concurrent users: median cached 234ms, 95th percentile 521ms; median uncached 687ms, 95th percentile 1,240ms; cache hit rate 79%. All values at the primary 100-user target satisfy specified response time requirements.

B. Throughput, Error Rate, and Data Freshness

System throughput reached 312 requests per second at the 100-user level and 487 requests per second at the 200-user level. The error rate at 100 users was 0.03%, within acceptable thresholds. Event propagation latency — from publication to persistence across all subscribing services — averaged 78ms at the 100-user level. For data changes in external systems propagated through scheduled synchronization, freshness latency ranged from 78 seconds (fastest-synchronized domains) to 62 minutes (least frequently synchronized), with no missed synchronization cycles observed over the seven-day evaluation period.

C. Availability, Resilience, and User Experience

System availability over the seven-day evaluation period was 99.71%, satisfying the 99.5% requirement. During fault injection testing, the system maintained availability for all features except those directly dependent on the terminated service, with full recovery within 18 seconds in all cases. The user experience study showed an 89% reduction in customer profile retrieval time (from 7.3 minutes to 48 seconds), an improvement in task accuracy from 89% to 98%, and perceived efficiency ratings improving from 2.1 to 6.1 on a seven-point scale. Improvements were consistent across all three role groups.

XIII. Deployment Architecture and Operations

The deployment architecture provides the resilience, scalability, and operational visibility required for a production e-commerce environment.

A. Container Orchestration and Continuous Delivery

Each microservice is packaged as a container image following the twelve-factor application methodology, ensuring portability and reproducibility. Images are stored in a private registry tagged with the producing commit identifier, providing full traceability from any running instance to its exact source version. Container orchestration manages service instance lifecycles with a minimum of two instances per service distributed across availability zones. Rolling update deployments replace instances one at a time with readiness checks, maintaining minimum capacity throughout. Automatic rollback triggers if a deployment raises the error rate above threshold. The continuous delivery pipeline progresses from commit through static analysis, unit testing, image build, integration testing, contract verification, and staging deployment, with a manual approval gate before production release.

B. Observability Framework

The observability framework provides three complementary views: metrics, logs, and distributed traces. Metrics — including request rate, error rate, response time percentiles, cache hit rate, database query duration, and event processing lag — are collected every fifteen seconds and visualized on real-time dashboards with automated alerting. Structured logs enriched with correlation identifiers enable cross-service request reconstruction. Distributed tracing captures timing and outcomes for every operation in the execution path of a request, visualized as waterfall diagrams that immediately reveal performance bottlenecks.

C. Disaster Recovery

The disaster recovery strategy targets a recovery time objective of thirty minutes and a recovery point objective of five minutes. The RTO is met through multi-instance deployment with automated failover and infrastructure-as-code reproducibility. The RPO is met through continuous database replication to a geographically separate standby, with replication lag monitored and alerted when it approaches two minutes. The event streaming platform retains events for a minimum of seven days, providing an additional recovery mechanism through event replay for database reconstruction following catastrophic failure.

XIV. Related Systems and Comparative Analysis

A comparative analysis of existing CRM solutions provides important context for evaluating the contributions of the proposed Unified CRM system. Commercially available CRM platforms, bespoke monolithic implementations, and prior microservices-based CRM prototypes each offer a different balance of capabilities, and understanding this landscape clarifies where the proposed system's design decisions add distinct value.

A. Commercial CRM Platforms

Market-leading commercial CRM platforms — including widely adopted cloud-based solutions positioned for enterprise retail — offer comprehensive out-of-the-box functionality covering customer profile management, campaign management, analytics dashboards, and integration connectors for common third-party systems. Their primary strength is breadth of functionality and the speed with which an organization can achieve a basic level of customer data consolidation, since little custom development is required for standard use cases.

However, commercial platforms present several limitations for organizations with complex, heterogeneous data landscapes. First, they are designed around a generic data model that may not align with the specific entities and relationships of a particular organization's customer management domain. Adapting the platform to accommodate domain-specific entities — such as the installment payment schedules and behavioral tagging taxonomy used in the proposed system — typically requires significant configuration or custom development using the platform's proprietary extension framework, introducing a dependency on the platform vendor's technology decisions and release cadence.

Second, commercial platforms are monolithic in their internal architecture, even when offered as cloud-hosted services. The inability to scale individual functional areas independently means that an organization experiencing high load on its order history queries must provision additional capacity for the entire platform, including modules under low load. Third, the data sovereignty implications of storing customer data in a third-party cloud platform require careful evaluation in jurisdictions with strong data protection regulations, adding compliance complexity that a self-hosted solution avoids.

Fourth, the integration capabilities of commercial platforms, while extensive for common third-party systems, may not support the specific external systems used by a given organization. Custom integration development in this context requires working within the constraints of the platform's extension framework rather than implementing the most appropriate integration pattern for the specific systems involved. For organizations with significant quantities of customer data in legacy or proprietary external systems — as is the case in the e-commerce context addressed by this paper — this limitation can be a significant barrier to achieving the level of data consolidation required.

B. Bespoke Monolithic CRM Implementations

Organizations that find commercial platforms insufficiently flexible or insufficiently aligned with their specific domain frequently invest in bespoke monolithic CRM implementations — custom-developed applications that consolidate customer data in a manner precisely tailored to the organization's data model and operational workflows. Bespoke monolithic systems offer the highest degree of customization and can implement exactly the data models, business rules, and user workflows required without the constraints of a commercial platform's extension framework.

However, bespoke monoliths accumulate the well-documented limitations of the monolithic architectural pattern over time. As the application grows in scope — incorporating new data sources, new user workflows, and new integration requirements — the codebase becomes increasingly difficult to modify safely. A change in one module requires comprehensive regression testing of the entire application, slowing the pace of evolution and increasing the cost of responding to new business requirements. Deployment of any change, however small, requires a full application release cycle, preventing the independent evolution and deployment of different functional areas.

Performance scaling is also constrained in a monolithic architecture. When a specific functional area — such as order history retrieval during a peak sales period — requires additional computational resources, the entire application must be scaled, consuming resources in functional areas that are not under load. This inefficiency becomes increasingly costly as the scale of the platform grows.

The Unified CRM system addresses these limitations while retaining the customization advantages of a bespoke implementation. By decomposing the application into independently deployable microservices, each bounded to a specific customer data domain, the system achieves the same degree of domain-specific customization as a bespoke monolith while enabling independent scaling, independent deployment, and independent evolution of each service. The trade-off is increased operational complexity — managing a fleet of microservices requires more sophisticated deployment infrastructure and observability tooling than managing a single

monolithic application — but the operational investment is justified by the architectural benefits at the scale and pace of change characteristic of large e-commerce organizations.

C. Prior Microservices CRM Prototypes

A small number of research prototypes and early industry implementations of microservices-based CRM systems have been reported in the literature. These prior works share the foundational architectural premise of the proposed system — decomposing CRM functionality into independently deployable services — but differ in their decomposition strategy, their approach to inter-service communication, and their handling of the data synchronization challenges that arise in heterogeneous data environments.

Several prior works adopt a coarser-grained decomposition than the proposed system, grouping related customer data domains into larger service boundaries to reduce the operational overhead of managing many small services. While this approach simplifies the deployment topology, it limits the granularity of independent scaling and increases the risk of a large service's internal complexity recreating the maintenance challenges of the monolith at a smaller scale. The proposed system's finer-grained decomposition — with separate services for customer information, membership, rewards, orders, and behavioral tags — reflects the specific access pattern and scalability requirements of each domain rather than a uniform granularity applied across the board.

Prior works that address inter-service communication predominantly adopt synchronous REST-based approaches, in which services call each other directly to retrieve data needed for a composite response. The proposed system's hybrid approach — synchronous parallel fan-out for read operations combined with asynchronous event streaming for write propagation — has not been previously reported in the CRM microservices literature. This design explicitly acknowledges that the latency requirements of interactive read operations and the consistency requirements of write operations call for different communication patterns, and provides a principled rationale for applying each pattern to the appropriate operation type.

D. Key Differentiators of the Proposed System

The proposed Unified CRM system distinguishes itself from both commercial platforms and prior microservices CRM implementations through four key design characteristics. First, the attribute-level, dual-enforcement role-based access control model — in which access rules are applied at both the API gateway and the frontend, with a centralized configuration serving as the single source of truth — provides a more comprehensive and consistent security posture than the coarser role-based restrictions typically implemented in commercial platforms or prior research prototypes.

Second, the event-driven cache invalidation strategy provides a data freshness guarantee that is qualitatively superior to the TTL-based approaches used in prior systems. By invalidating cache entries within milliseconds of the triggering data change event rather than waiting for the TTL to expire, the system provides a near-real-time customer view that is essential for high-interaction e-commerce environments.

Third, the probabilistic record linkage component of the data synchronization subsystem addresses the customer identity resolution problem that is frequently encountered in practice but rarely addressed in prior CRM microservices literature. The ability to identify and merge records representing the same customer across heterogeneous external systems is a prerequisite for the unified customer view that is the primary deliverable of any CRM platform, and the absence of this capability is a significant limitation of prior implementations.

Fourth, the operational observability framework — combining metrics, structured logs, and distributed tracing with automated alerting and anomaly detection — provides the production-grade operational visibility required to operate a system of this complexity in a real-world e-commerce environment. The importance of observability in microservices systems is well-recognized in the practitioner literature but rarely demonstrated in research implementations, which typically focus on functional correctness without addressing the operational lifecycle of the deployed system.

XV. Broader Impact and Applicability

While the Unified CRM system was designed for the specific context of e-commerce customer management, the architectural patterns and implementation strategies described in this paper have broader applicability to a range of domains in which multiple independent systems must be integrated to provide a unified view of a shared entity — whether that entity is a customer, a patient, a student, or an asset.

A. Healthcare Patient Data Consolidation

Healthcare organizations face a data fragmentation challenge structurally similar to that addressed by the proposed CRM system: patient records are distributed across electronic health record systems, laboratory information systems, imaging archives, pharmacy systems, and billing platforms, each using different identifiers and data models for the same patient. The microservices architecture, event-driven integration, probabilistic record linkage, and role-based access control patterns described in this paper are directly applicable to the patient data consolidation problem, with the additional requirement for more stringent compliance with data protection regulations such as HIPAA in the United States or applicable national health data protection frameworks in other jurisdictions.

B. Financial Services Customer Intelligence

Financial services organizations — banks, insurance providers, and wealth management firms — maintain customer data across product silos: retail banking, mortgage lending, insurance policies, and investment accounts may each be managed by separate systems with limited cross-system data sharing. A unified customer intelligence platform built on the architectural foundations described in this paper would enable relationship managers to access a comprehensive view of a customer's financial relationship with the institution, supporting more effective needs assessment, more personalized product recommendations, and more proactive risk management.

C. Educational Institution Student Records

Educational institutions managing student records across admissions, academic performance, financial aid, housing, library, and extracurricular activity systems face a fragmentation challenge analogous to the e-commerce CRM context. A unified student record platform built on the proposed architecture would enable academic advisors, financial aid officers, and student support staff to access a complete view of a student's institutional engagement, supporting earlier identification of students at risk of academic difficulty or financial hardship, and enabling more coordinated institutional responses.

D. Transferability of Architectural Patterns

The core architectural patterns of the proposed system — microservices decomposition by bounded context, event-driven integration through a distributed message broker, probabilistic entity resolution for cross-system identity matching, hybrid synchronous-asynchronous communication, and dual-enforcement role-based access control — are domain-agnostic. They address fundamental challenges that arise whenever multiple independent systems must be integrated to provide a unified view of a shared entity, regardless of the specific domain or the specific systems involved. Organizations in any domain facing these challenges can adopt the architectural framework described in this paper and adapt the specific service decomposition, data models, and synchronization strategies to their particular context.

The operational considerations described in this paper — the container-based deployment model, the continuous delivery pipeline, the observability framework, and the disaster recovery strategy — are similarly transferable. The maturity of the cloud-native tooling ecosystem means that the infrastructure described in this paper is accessible to organizations of a wide range of sizes, not only large enterprises with substantial dedicated engineering teams. The operational framework represents current best practice for production-grade microservices deployment, and its adoption alongside the architectural patterns described in this paper provides a complete blueprint for organizations beginning their journey from fragmented, legacy data systems to a unified, scalable, and operationally excellent data platform.

XVI. REFERENCES

[1] Z. Soltani and N. J. Navimipour, "Customer Relationship Management Mechanisms: A Systematic Review of the State-of-the-Art Literature and Recommendations for Future Research," Computers in Human Behavior, Elsevier, vol. 61, pp. 667-688, 2016.

[2] C. Marcinkevage and A. Kumar, "Customer Relationship Management: A Systematic Framework for a Successful Integration," Journal of Business Research, Elsevier, vol. 199, Article 115531, 2025.

[3] N. Alshuqayran, N. Ali, and R. Evans, "A Model-Driven Architecture Approach for Recovering Microservice Architectures: Defining and Evaluating MiSAR," Information and Software Technology, Elsevier, 2025.

[4] H. M. Ayas, R. Hebig, and P. Leitner, "An Empirical Investigation on the Competences and Roles of Practitioners in Microservices-Based Architectures," Journal of Systems and Software, Elsevier, 2024.

[5] C. Ledro, N. Dalla Pozza, and A. Borghini, "Artificial Intelligence in Customer Relationship Management: A Framework for Successful Integration," Journal of Business Research, Elsevier, 2025.

[6] G. Narkhede, N. Shapira, and T. Palino, Kafka: The Definitive Guide - Real-Time Data and Stream Processing at Scale, O'Reilly Media, 2nd ed., 2021.

[7] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2nd ed., 2021.

[8] R. F. Muhammad, N. Hartmann, L. Linsbauer, and A. Wortmann, "Re-Engineering Microservices into Web-Based Software Product Lines," in Proc. SPLC-B '25, ACM, 2025.

[9] M. Richardson, Microservices Patterns: With Examples in Java, Manning Publications, 2018.

[10] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

[11] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual Understanding of Microservice Architecture: Current and Future Directions," ACM SIGAPP Applied Computing Review, vol. 17, no. 4, pp. 29-45, 2017.

[12] W. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, Microservice Architecture: Aligning Principles, Practices, and Culture, O'Reilly Media, 2016.

[13] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect's Perspective, Addison-Wesley, 2015.

[14] P. Kaur and A. Bhardwaj, "In-Memory Data Caching for High-Performance Web Applications," International Journal of Computer Applications, vol. 180, no. 30, pp. 1-5, 2018.

[15] A. Gupta and R. Singh, "Comparative Study of SQL and NoSQL Databases for Retail Data Management," Journal of Database Management, vol. 32, no. 3, pp. 45-60, 2021.