|  |
| --- |
| **C2C Marketplace** |
| Architecture Design Document |
|  |
|  |
| **2025-09-25** |
| **Nikhil Gupta** |

This document is an Architecture Design Document for developing **C2C Marketplace**.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | 2025-09-25 | Nikhil Gupta | Initial document creation |
| 0.2 | 2025-10-11 | Nikhil Gupta | Pre-final Report Submission |
| 0.3 | 2025-10-19 | Nikhil Gupta | Changes according to Feedback |
| 0.4 | 2025-10-26 | Nikhil Gupta | Final Report Submission |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. Overview 3](#_Toc516321189)

[2. Requirements 7](#_Toc516321190)

[2.1. Functional Requirements 7](#_Toc516321191)

[2.2. Non-functional Requirements 12](#_Toc516321192)

[2.3. Quality Attributes 13](#_Toc516321193)

[3. Architecture 14](#_Toc516321194)

[4. Modules 15](#_Toc516321195)

[Appendix 16](#_Toc516321196)

[A. Domain Model 17](#_Toc516321197)

[B. Quality Scenarios 18](#_Toc516321198)

[C. Quality Scenario Analysis 19](#_Toc516321199)

[D. Candidate Architectures 20](#_Toc516321200)

[E. Candidate Architecture Evaluation 21](#_Toc516321201)

[F. Architecture Design 22](#_Toc516321202)

[G. Architecture Evaluation(ATAM) 23](#_Toc516321203)

# Overview

// A1. System Definition

// C1-1. Is the defined system boundary clear?

// C1-2. Is there sufficient explanation of System’s operation and business environment as business drivers?

* 1. **Introduction**

A Customer-to-Customer (C2C) Marketplace is a platform that allows users to **buy and sell** both used and new products. It provides a digital space where individuals can easily list their items and buyers can discover items they want using advanced search. The Marketplace should ensure safety and security for all the operations that happen in the system.

* 1. **System Definition**

The purpose of this project is to design a Customer-to-Customer (C2C) Marketplace.

Figure 1 below depicts the System Boundary and how C2C Marketplace will interact with the outside Components.

* System will provide interface for Actors like Buyer, Seller & Moderators.
* System will interact with an external Notification Service Interface & External LLM.

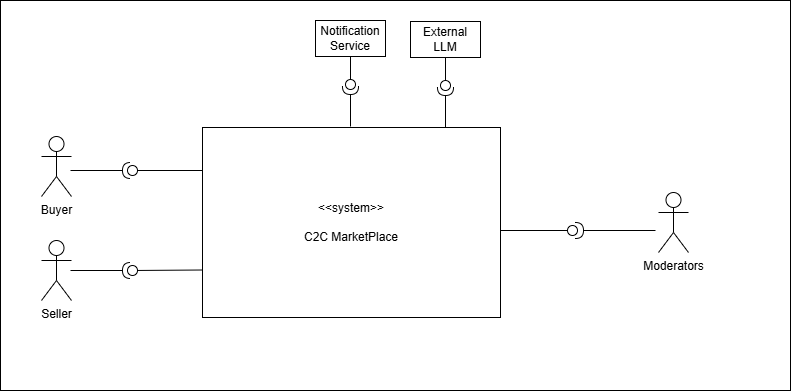


Figure 1: **System boundary for C2C Marketplace**

* 1. **Business Context & Drivers**

In a modern C2C marketplace, primary business challenge is to create a trusted, cohesive ecosystem where the user journey from discovery to final purchase is not only possible, but also intuitive and secure.

To achieve this, our architecture will be guided by these core principles:

* **Easy Discovery**: We will help buyers find what they want quickly. This will be done through a simple, effective search feature and by showing personalized recommendations on the home page.
* **Seamless Listing Experience**: We will make it effortless for sellers to list their items. The process will be smooth and guided, with helpful features like auto-saving drafts and a final preview before publishing.

# Requirements

## Functional Requirements

// A2. Functional Requirement Specification

// C2-1. Is there sufficient functional requirement specification to affect System’s architecture?

// C2-2. Is the relationship between use cases clear?

// C2-3. Is the division of use cases explicit?

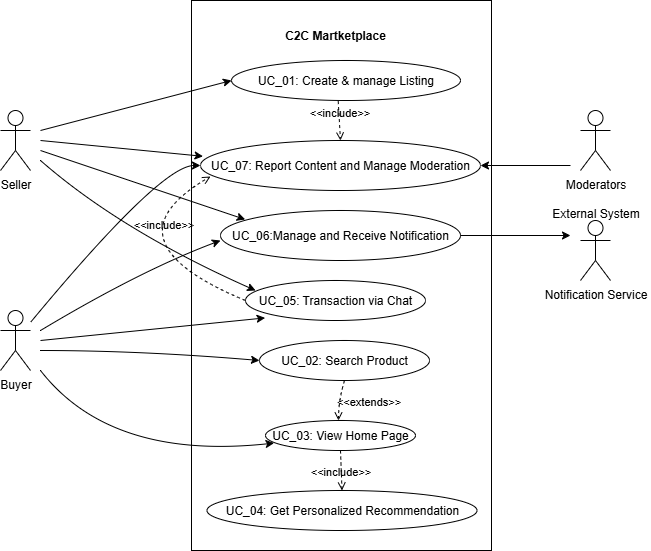


Figure 2: C2C Marketplace - Use Case Diagram

|  |  |
| --- | --- |
| **UC\_01** | Create & Manage Listing |
| Description | User can Create a new product listing & Update or Delete an existing product listing. |
| Actor | User (Seller) |
| Pre-condition | The User must be signed-in and have an authorized account to create a listing. |
| Post-condition | **On Success**: A new listing is created in the system.  **On Failure**: The listing is not created, and the Seller is notified of the reason for the failure. |
| Basic Flow | 1. User requests the system to create a new item listing.  2. User uploads images and enters common item details (like Product Category, Title, Description, Price, and Location. The user can also specify if it is a premium listing (Y/N). (AF1)  3. System **dynamically determines and presents** additional fields relevant to the selected product category, utilizing a **configurable attribute schema** to support diverse item types and future category additions. For example:   * If user selects the **"Cars"** category, the system presents fields for Make, Model, Year, and Mileage. * If the user selects the **"Mobile Phones"** category, the system instead presents fields for Brand, Storage Capacity, and Battery Health.   4. User then completes these category-specific fields.  5. System runs content moderation checks on all the filled-in data and confirms it passes the validation. (AF2)  6. System presents a preview of the listing showing the user how their listing will appear to other users.  6. User reviews and validates the data in the preview, then publishes the listing.  7. System confirms that the new listing has been successfully created. |
| Additional Flow | AF1. Auto Save Draft: While the User is entering details (during steps 2-3), the system automatically saves the progress as a draft. This allows the user to restore the last saved version if interrupted.  AF2: Content Moderation Failure:  1. If the system's content moderation check detects a policy violation, the listing process is paused.  2. System indicates the specific issue to the Seller and offers potential solutions.  3. User can then either correct the details & resubmit the listing information, or they may be given an option to submit the listing for a manual review. |

|  |  |
| --- | --- |
| **UC\_02** | Search Product |
| Description | User can search for an Item by entering keywords & system returns the results by using semantic search on entered Keywords and location. |
| Actor | User (Buyer) |
| Pre-condition | 1. Listings are already created and are indexed  2. User location is available |
| Post-condition | Results returned with categorical breakdowns |
| Basic Flow | 1. User submits a search request to the system with keywords and optional filters.  2. System validates the request and executes a query against the dedicated search index, using keyword matching, semantic search, and filters to retrieve initial listings ranked by a core relevance score.  3. System retrieves supplementary data (e.g., listing date, seller rating) for the initial results and applies defined business rules to adjust the ranking based on factors like:   * Geographic Proximity * Listing Recency * Seller Reputation * Listing Type (e.g., premium)   4. System identifies relevant advertisements and merges them into the re-ranked listing results according to placement strategies.  5. System prepares and returns the final, merged, and ranked list (including pagination information) to the user. (AF1) |
| Additional Flow | AF1: No Results Found: This flow occurs at Step 5 if no listings match the Buyer's search criteria. The system communicates, "No results found".  To be helpful, the system also suggests alternative actions, such as checking for spelling errors, removing filters, or it may present popular items from a similar category. |

|  |  |
| --- | --- |
| **UC\_03** | View Home Page |
| Description | This use case describes what happens when a user initiates a session with the system’s main interface, which is then loaded along with recommendations for the user |
| Actor | User |
| Pre-condition | Internet Connection should be present. |
| Post-condition | On success, the system will assemble and provide the main interface along with the recommendation data. |
| Basic Flow | 1. User initiates a session with the main interface.  2. System identifies the authenticated user and initializes their session.  3. System fetches personalized content feeds for the Buyer, such as "Recommended For You," "New Items from Sellers You Follow," and "Recently Viewed." (UC\_04) (AF1)  4. System also fetches general content like "Trending Near You" and featured categories.  5. System assembles and presents the main interface content, prioritizing personalized content and ensuring search feature is readily accessible. |
| Additional Flow | AF1: New or Guest User, System has no personalization data. Instead of a personalized feed, it will return a generic feed consisting of trending items, location-based bestsellers, or recently listed Items. |

|  |  |
| --- | --- |
| **UC\_04** | Get Personalized Recommendations |
| Description | This use case describes how the system provides proactive, context-aware product recommendations to a user in various situations, such as when they are accessing the main interface or viewing details of a specific product. |
| Actor | User (Buyer) |
| Pre-condition | User is logged in. |
| Post-condition | System presents a set of relevant personalized items to the user. |
| Basic Flow | Scenario A: Recommendations on the Home Page  1. Trigger: User initiates a session with the main interface.  2. System analyzes the Buyer's profile, recent activity (views, searches), and location to understand their intent.  3. System generates and presents several personalized feeds, such as "Recommended For You," "Because You Viewed X," and "New Items Nearby." (AF1)  4. System also considers relevant targeted ads in the system and creates a final curated list of recommended items.  5. System displays the recommendations to the user.  Scenario B: Product Detail Page Recommendations  1. Trigger: User requests details for a specific product.  2. System analyzes the current product's attributes (category, brand, price).  3. System generates and presents contextual recommendations designed to increase conversion, such as "Similar Items" (alternatives at different price points) and "Complementary Items" (accessories or related products).  4. System presents these related Items to the user. |
| Additional Flow | AF1. Cold start: For new users, as a fallback, system uses popular listings of user’s location. |

|  |  |
| --- | --- |
| **UC\_05** | Secure Chat Messenger |
| Description | This use case describes how a User (a Buyer and a Seller) communicate in real-time through a secure messaging feature. It allows the Buyer to ask questions about an item and for both parties to negotiate a price by making formal offers and counter-offers, leading to a sales agreement. |
| Actor | User (Buyer & Seller) |
| Pre-condition | 1. Users are logged in.  2. A product listing must be active and available in the system. |
| Post-condition | 1. On success, an offer is accepted, the system creates a formal order, and the listing is marked as sold.  2. On failure, the negotiation ends without an agreement.  3. The chat history is saved for future reference. |
| Basic Flow | 1. User navigates to a product listing and initiates a chat.  2. System establishes a secure communication channel between the users.  3. Users can exchange text messages to discuss the item details.  4. After discussions, user submits a formal offer with a specific price.  5. System presents this offer to the other user.  6. The other user accepts the offer. (AF1)  7. This flow (steps 2-4) continues until an offer is accepted or one of the users ends the conversation.  8. System confirms the agreement to both the users, and marks the items as sold. |
| Additional Flow | AF1: Counter Offer: At Step 6, instead of accepting, the user can decline the offer and submit a counter-offer with a different price. The flow then returns to Step 5, with the Buyer now having the option to accept or decline.  AF2: Automated Content Filtering: If a message contains content that violates policies, such as requests for off-platform payments or personal information, System automatically blocks the message to prevent frauds.  AF3: Spam Prevention: System monitors message frequency and will temporarily limit a user's ability to send messages if they are suspected of spam or abuse. |

|  |  |
| --- | --- |
| **UC\_06** | Manage and Receive Notifications |
| Description | This use case describes how a user customizes their notification settings to control how and when they receive alerts from the system. It also covers how a user interacts with a received notification to navigate directly to relevant content within the application |
| Actor | User (Buyer/Seller) |
| Pre-condition | User account exists with verified contact details (email & contact number) to receive notifications. |
| Post-condition | 1. The user's notification preferences are saved and applied to all future alerts.  2. Upon interacting with a notification, the user is successfully directed to the correct screen. |
| Basic Flow | 1. User accesses the notification settings feature within the account settings.  2. System presents all notification categories along with the available channels (e.g., Push, In-App, Email, SMS) for each.  3. User adjusts the settings to their preference and saves the changes.  4. System confirms that the preferences have been successfully updated and will be used in future |
| Additional Flow | AF1: Interacting with a Received Notification:  1. A user receives an alert (e.g., a push notification on their device) that an event has occurred in the system.  2. User taps on the notification.  3. System opens the application and directs the user to the relevant section. (e.g., to the specific chat for a new message). |

## Non-functional Requirements

// A5. Quality Requirement Specification

// C5-2. Is the specification of quality requirements appropriate?

// C5-3. Is quality requirement measurable?

// C5-4. Is the allowance of non-functional requirement clear?

|  |  |  |
| --- | --- | --- |
| **NFR\_01** | **Performance** | **Search operation response time should be low** |
| Description | Searching of an item using user input along with filters and user location should be low | |
| Environment | During peak operational hours, with the system handling a high volume of concurrent users. | |
| Stimulus | A user submits a search query with optional filter | |
| Response | 1. System receives the GET /search request at the API Gateway. 2. The Search Service validates the query and filter parameters (e.g., location, category, price range). 3. Search Service executes the compiled query against the dedicated Search Index cluster. 4. Search Service receives the list of matching listing IDs from the index. 5. Search Service optionally re-ranks the results based on business logic (e.g., boost new sellers, seller rating). 6. System returns the final, paginated list of listings to the client. | |
| Measure | [Response Time] = [time when final ordered list is returned] – [time of user request] | |
| Allowance | [Response Time] <= N milliseconds | |

|  |  |  |
| --- | --- | --- |
| **NFR\_02** | **Performance** | **Home Screen Loading time** |
| Description | The maximum time allowed for the backend to aggregate and deliver all the data required to render the user's home screen, including personalized content | |
| Environment | Under normal operational conditions when a user opens the application. | |
| Stimulus | User opens the app/home route; client requests home payload. | |
| Response | 1. System receives the request, identifies the user, and initiates calls to various downstream services (e.g., recommendations, listings) to retrieve the necessary content feeds. 2. System receives the GET /home request at the API Gateway. 3. The Home Controller identifies the user and initiates *parallel* calls to downstream services (e.g., Recommendation Service, Listing Service for categories). 4. Home Controller waits for all critical downstream services to respond or time out. 5. System aggregates the content from all responses into a single JSON data payload. 6. System returns the complete data payload to the user's client. | |
| Measure | [Response Time] = [time when data payload is returned] – [time when request is initiated]. | |
| Allowance | [Response Time] <= N milliseconds | |

|  |  |  |
| --- | --- | --- |
| **NFR\_03** | **Performance** | **Service Scalability** |
| Description | System should automatically adjust its resource allocation to handle significant fluctuations in traffic, maintaining performance without manual intervention. | |
| Environment | During a period of rapidly increasing user traffic, such as a marketing event or flash sale. | |
| Stimulus | The incoming request rate for key services increases by N% over a 15-minute period. | |
| Response | 1. The cloud monitoring infrastructure component of the system detects that there is increase in key service metrics (e.g. request count) has increased by N% over a 15-minute period. 2. Cloud monitoring infrastructure detects that a key metric (e.g., CPU utilization) has breached the pre-defined auto-scaling threshold. 3. The orchestrator (e.g., Kubernetes HPA or an Auto-Scaling Group) triggers a scale-out event. 4. System provisions one or more new service instances (pods/containers). 5. The new instances start, pass their mandatory health checks, and register with the load balancer. 6. The load balancer begins routing live traffic to the new, healthy instances. | |
| Measure | [Time to Scale] = [time when new provisioned infra is able to accept traffic] - [time when system detects the increase by N% over a 15-minute period] | |
| Allowance | [Time to Scale] <= N seconds | |

|  |  |  |
| --- | --- | --- |
| **NFR\_04** | **Reliability** | System Recovery |
| Description | System should be able to automatically recover from a critical component failure and restore full service functionality without requiring manual intervention | |
| Environment | A critical, stateless service instance crashes, or a primary node of a stateful service (like a database) fails. | |
| Stimulus | A critical, stateless service instance (e.g., ListingService) crashes or fails its health check. | |
| Response | 1. The system's orchestrator (e.g., Kubernetes) detects the instance is unhealthy and terminates it.  2. The load balancer immediately stops routing traffic to the failing instance.  3. The orchestrator's replication controller detects the replica count is below the desired state.  4. The orchestrator immediately schedules a replacement instance.  5. A new, healthy instance is started, passes its health checks, and begins accepting traffic, restoring full service capacity. | |
| Measure | [Time to recover] = [time when a new instance begins accepting traffic] – [time when failure is detected] | |
| Allowance | [Time to recover] <= 'N' minutes | |

## Quality Attributes

// A5. Quality Requirement Specification

// C5-2. Is the specification of quality requirements appropriate?

// C5-3. Is quality requirement measurable?

|  |  |  |
| --- | --- | --- |
| **QA\_01** | **Usability** | Search Result Relevance |
| Description | This attribute measures the effectiveness of the search algorithm by quantifying how often the displayed results are relevant enough to warrant user engagement. | |
| Environment | In the live production environment where users are actively searching for items. | |
| Stimulus | A user enters a search query and system returns an ordered list of results. | |
| Response | The user reviews the list, finds the top results relevant to their intent, and clicks on a result to view the item's detail page.   1. System displays the paginated list of search results to the user. 2. The user visually scans the *first page* of results. 3. The user identifies a relevant item within the top N (e.g., 5) results. 4. The user clicks on that item to view its detail page. | |
| Measure | [Search Result Click-Through Rate] (CTR) = (Total Searches that Result in a Click) / (Total Number of Searches Performed) | |

|  |  |  |
| --- | --- | --- |
| **QA\_02** | **Usability** | Recommendation Usefulness |
| Description | This attribute measures the effectiveness of the recommendation system by quantifying user engagement with the items it suggests. | |
| Environment | In the live production environment with a diverse set of real users. | |
| Stimulus | A user is shown a carousel of recommended items on the home page or a product detail page. | |
| Response | The user finds the recommendations relevant and interesting, leading them to click on one or more of the suggested items to learn more.   1. System renders the "Recommended For You" carousel on the home page. 2. The user scrolls or visually scans the items in the carousel. 3. The user identifies a compelling item in the list. 4. The user clicks on the recommended item to view its detail page. | |
| Measure | [Recommendation Click-Through Rate] (CTR) = (Total Clicks on Recommended Items) / (Total Impressions of Recommended Items) | |

|  |  |  |
| --- | --- | --- |
| **QA\_03** | **Security** | Personal Information & Off-platform Solicitation Protection |
| Description | This security attribute ensures that users are proactively protected from frauds and privacy violations within the chat feature by automatically filtering prohibited content. | |
| Environment | During a real-time chat session between any two users. | |
| Stimulus | A user attempts to send a chat message containing sensitive information (e.g., a phone number) or a link to an external payment site. | |
| Response | The system's content moderation service analyzes the message in real-time, identifies the policy violation, blocks the message from being delivered, and notifies the sender. | |
| Measure | [Detection Accuracy] = (Correctly Identified Violations) / (Total Actual Violations)  [False Positive Rate] = (Incorrectly Blocked Legitimate Messages) / (Total Legitimate Messages Analyzed) | |

|  |  |  |
| --- | --- | --- |
| **QA\_04** | **Security** | Listing Moderation & Malicious Content Prevention |
| Description | This trust and safety attribute ensures that all new and updated listings are automatically scanned for prohibited or malicious content before they are made visible to buyers. | |
| Environment | During the listing creation or editing workflow. | |
| Stimulus | A seller submits a new listing that includes policy-violating text (e.g., prohibited items) or images (e.g., unsafe content). | |
| Response | The automated moderation service analyzes the listing content, detects the violation, rejects the submission, and provides specific feedback to the seller on how to correct the issue. | |
| Measure | [False Positive Rate] = (Incorrectly Blocked Legitimate Listings) / (Total Legitimate Listings Submitted) | |

|  |  |  |
| --- | --- | --- |
| **QA\_05** | **Performance** | Listing Visibility Latency |
| Description | This attribute defines the maximum acceptable delay from when a seller creates or updates a listing to when that change is reflected and visible in user-facing systems like search and recommendations. | |
| Environment | During normal system operations. | |
| Stimulus | A seller successfully publishes a new product listing or saves an update to an existing one. | |
| Response | The system persists the change to the primary database and successfully propagates the update to the search index and recommendation data stores, making the listing discoverable by buyers. | |
| Measure | [Latency] = [t\_visible\_in\_search] – [t\_published/t\_edited] | |

|  |  |  |
| --- | --- | --- |
| **QA\_06** | **Maintainability** | Adaptability for New Recommendation Algorithms |
| Description | This attribute defines the system's modifiability, ensuring that new recommendation or search algorithms can be integrated & deployed with minimal engineering effort and without requiring major architectural changes. | |
| Environment | During a planned development cycle in a staging/testing environment. | |
| Stimulus | The data science team provides a new, trained recommendation model that needs to be integrated into the production system. | |
| Response | An engineer integrates the new model by implementing a pre-defined service interface, allowing it to be deployed alongside the existing model for A/B testing without altering the core services that request recommendations. | |
| Measure | [Engineering Effort] = The total person-weeks required to integrate, test, and deploy a new recommendation algorithm. | |

|  |  |  |
| --- | --- | --- |
| **QA\_07** | **Security** | Real-time Seller Risk Scoring |
| Description | This fraud prevention attribute defines the system's ability to assess the risk level of a seller in real-time based on their actions, in order to flag potentially fraudulent activity early. | |
| Environment | A seller performs a significant action, such as creating an account or listing a high-value item. | |
| Stimulus | A new seller with no transaction history lists a high-demand electronic item at a price significantly below the market average. | |
| Response | The real-time risk engine processes signals associated with the seller and the listing, calculates a high-risk score, and automatically flags the listing for mandatory manual review before it is published. | |
| Measure | [Risk Scoring Latency] = [t\_score\_available] – [t\_trigger\_event] | |

|  |  |  |
| --- | --- | --- |
| **QA\_08** | **Response Time** | Listing Media Processing Time |
| Description |  | |
| Environment |  | |
| Stimulus |  | |
| Response |  | |
| Measure |  | |

|  |  |  |
| --- | --- | --- |
| **QA\_09** | **Performance** | Chat Message Delivery Time |
| Description | This attribute defines the end-to-end latency for a message sent between two users. A low latency is critical to ensure the chat feels conversational and real-time. | |
| Environment | During a chat session between two online users under normal network conditions. | |
| Stimulus | A user sends a message to another user. | |
| Response | The system receives the message from the sender, processes it, and delivers it to the recipient's client application. | |
| Measure | [Delivery Latency] = [t\_received] – [t\_sent]. | |

|  |  |  |
| --- | --- | --- |
| **QA\_10** | **Reliability** | Monitoring and Alerting |
| Description | This attribute ensures that key health metrics of all production micro services are actively monitored and that automated alerts are triggered when anomalies are detected. | |
| Environment | The system is running in the 24/7 production cloud environment. | |
| Stimulus | A critical backend service (e.g., Order Service) experiences a sustained spike in its server-side error rate (HTTP 5xx errors) exceeding a predefined threshold. | |
| Response | The monitoring system detects the threshold breach and automatically generates and sends a high-priority alert to the designated on-call engineering channel | |
| Measure | [Detection-to-Alert Latency] = [t\_alert\_sent] – [t\_threshold\_breached] | |

# Architecture

// A8. Architecture Documentation

// C8-1. Is allocation of processes, etc. appropriate? (deployment)

// C8-2. Is grouping appropriate in terms of components? (component & connector)

// C8-3. Is the description of System architecture appropriate?

# Modules

// A9. Module Specification

// C9-1. Is component specification sufficient to develop?

// C9-2. Is grouping appropriate in terms of module?

// C9-3. Is it appropriate to design dependencies between modules?

// C9-4. Is the work assignment appropriate?

Appendix

[A. Domain Model 17](#_Toc516321204)

[B. Quality Scenarios 18](#_Toc516321205)

[C. Quality Scenario Analysis 19](#_Toc516321206)

[D. Candidate Architectures 20](#_Toc516321207)

[E. Candidate Architecture Evaluation 21](#_Toc516321208)

[F. Architecture Design 22](#_Toc516321209)

[G. Architecture Evaluation(ATAM) 23](#_Toc516321210)

1. Domain Model

// A3. Domain Model Design

// C3-1. Is domain model sufficiently sub-divided?

// C3-2. Does domain model reflect architecture decisions?

**Entity-Control-Boundary** pattern is used to describe the conceptual model of C2C Market Place.

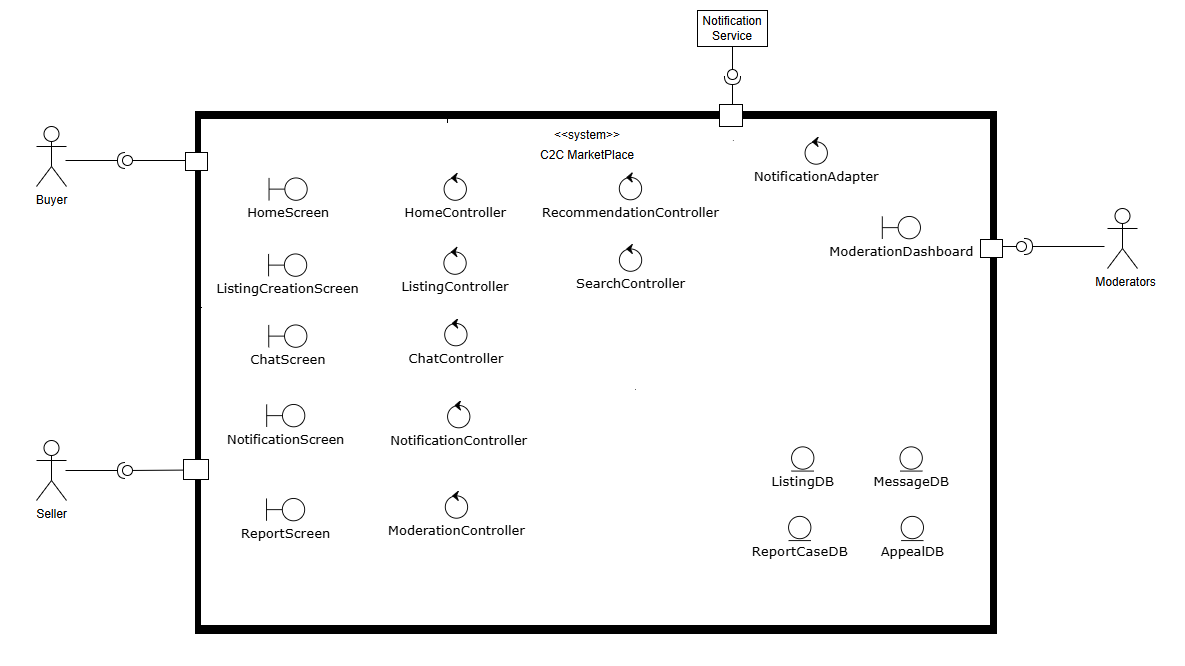


Figure 10: Domain Model for C2C Market Place

### Boundaries Components: These are the interfaces that users interact with directly.

* **HomeScreen**: The main landing screen that displays content and includes search and recommendation features.
* **ListingCreationScreen**: The UI where a Seller creates, previews, and manages their product listings.
* **ChatScreen**: The interface for sending and receiving messages and offers between users.
* **NotificationScreen**: The interface for managing notification preferences.
* **ReportScreen**: The UI used by any user to report suspicious content or other users.
* **ModerationDashboard**: A specialized UI for Moderators to review and act on reported cases and appeals.

### Controllers Components: These are the components containing the core business logic for each use case.

* **ListingController**: Manages the logic of creating, validating, and publishing a product listing.
* **SearchController**: Handles the logic of executing a search query and returning relevant results.
* **HomeController**: Orchestrates the assembly of the home page, including fetching recommendations.
* **RecommendationController**: Contains the logic for generating personalized recommendations for users.
* **ChatController**: Manages the business rules for messaging, offers, and finalizing agreements.
* **NotificationController**: Manages the logic for when and how to send notifications to users based on system events.
* **ModerationController**: Handles the workflow for automated and manual content moderation, including user reports and appeals.
* **NotificationAdapter**: A specialized controller that translates internal notification requests into the format required by external services.

### Entities: These are the core data objects that are persisted on the server.

* **ListingDB**: Represents a product or item that a Seller has put up for sale.
* **MessageDB**: Represents a single communication sent between a Buyer and a Seller.
* **ReportCaseDB**: An entity created to track a user-reported issue.
* **AppealDB**: Represents a user's request to have a moderation decision reviewed manually.

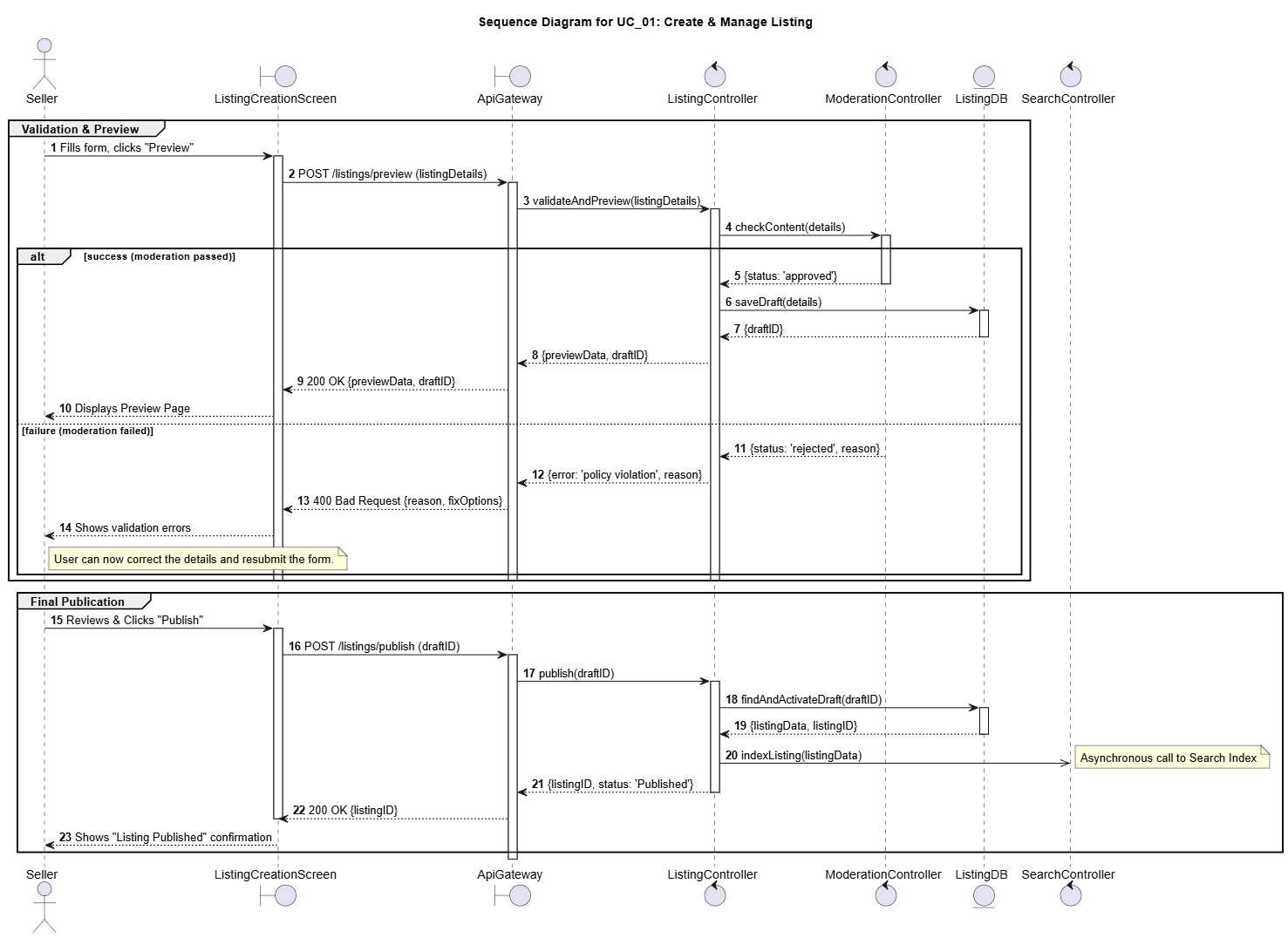


Figure 11: Sequence Diagram for UC01: Create & Manage Listing

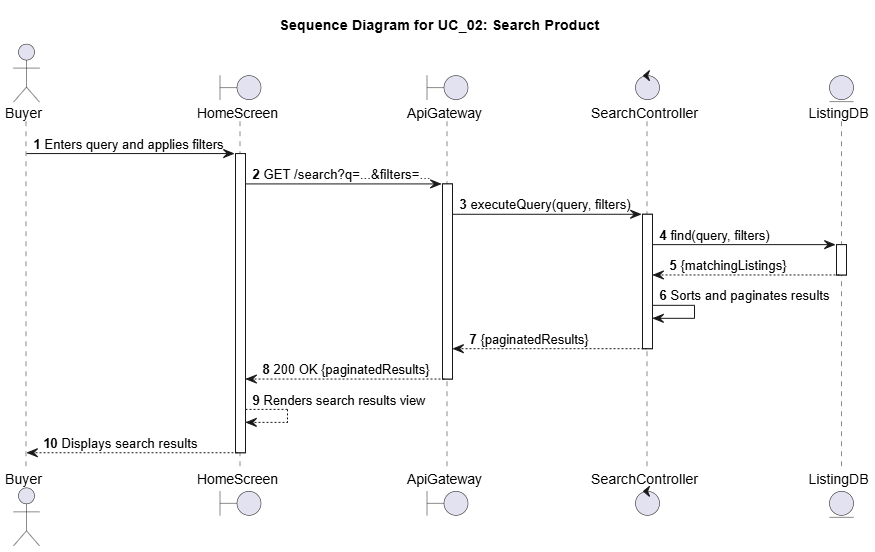


Figure 12: Sequence Diagram for UC02: Search Product

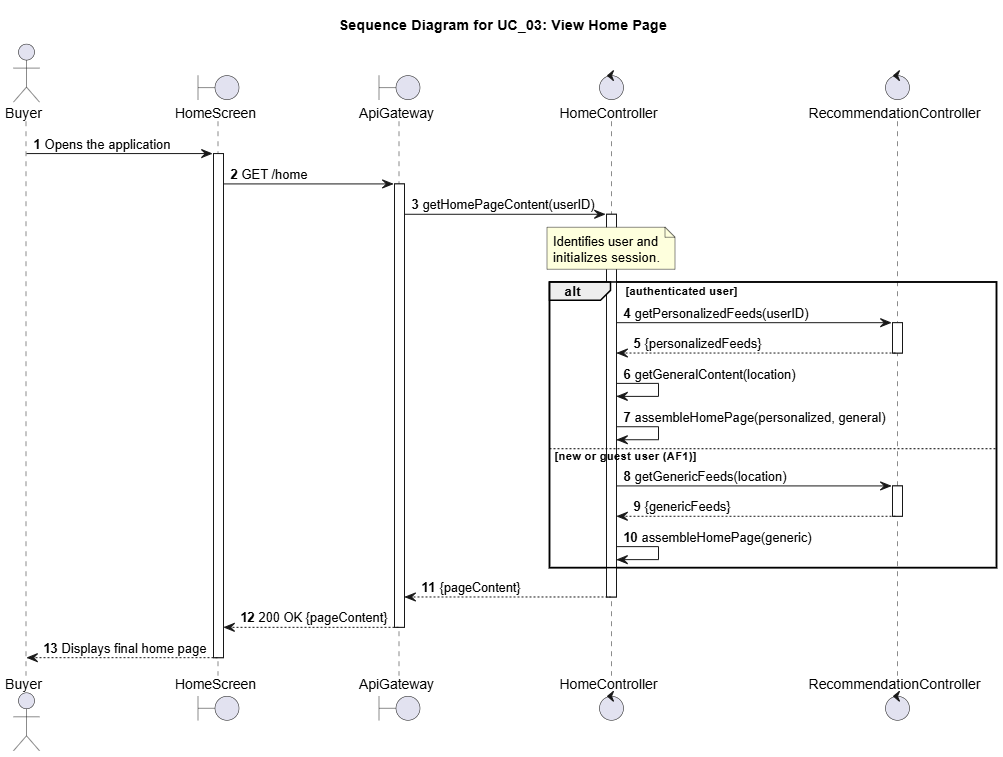


Figure 13: Sequence Diagram for UC03: View Home Page

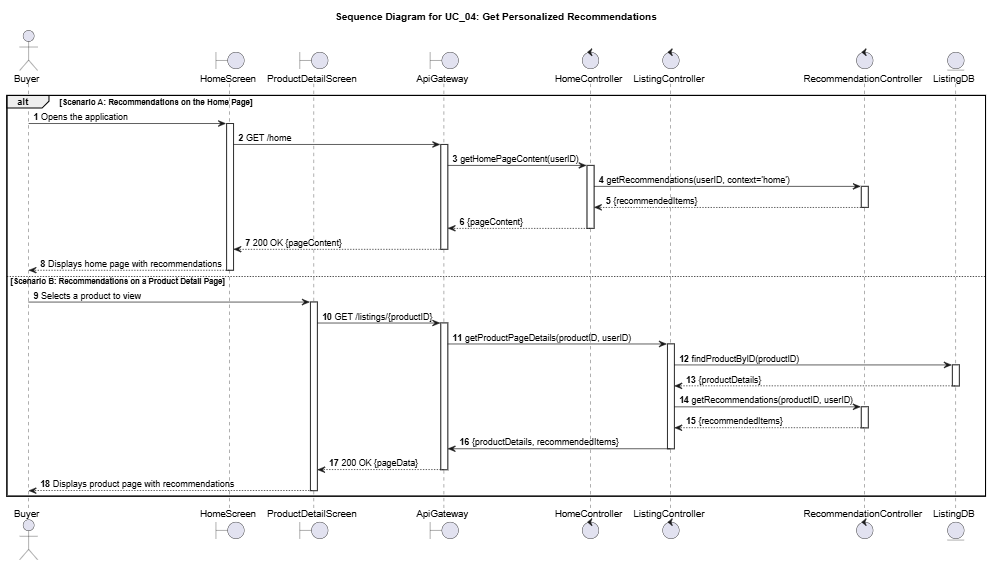


Figure 14: Sequence Diagram for UC04: Get Personalized Recommendations

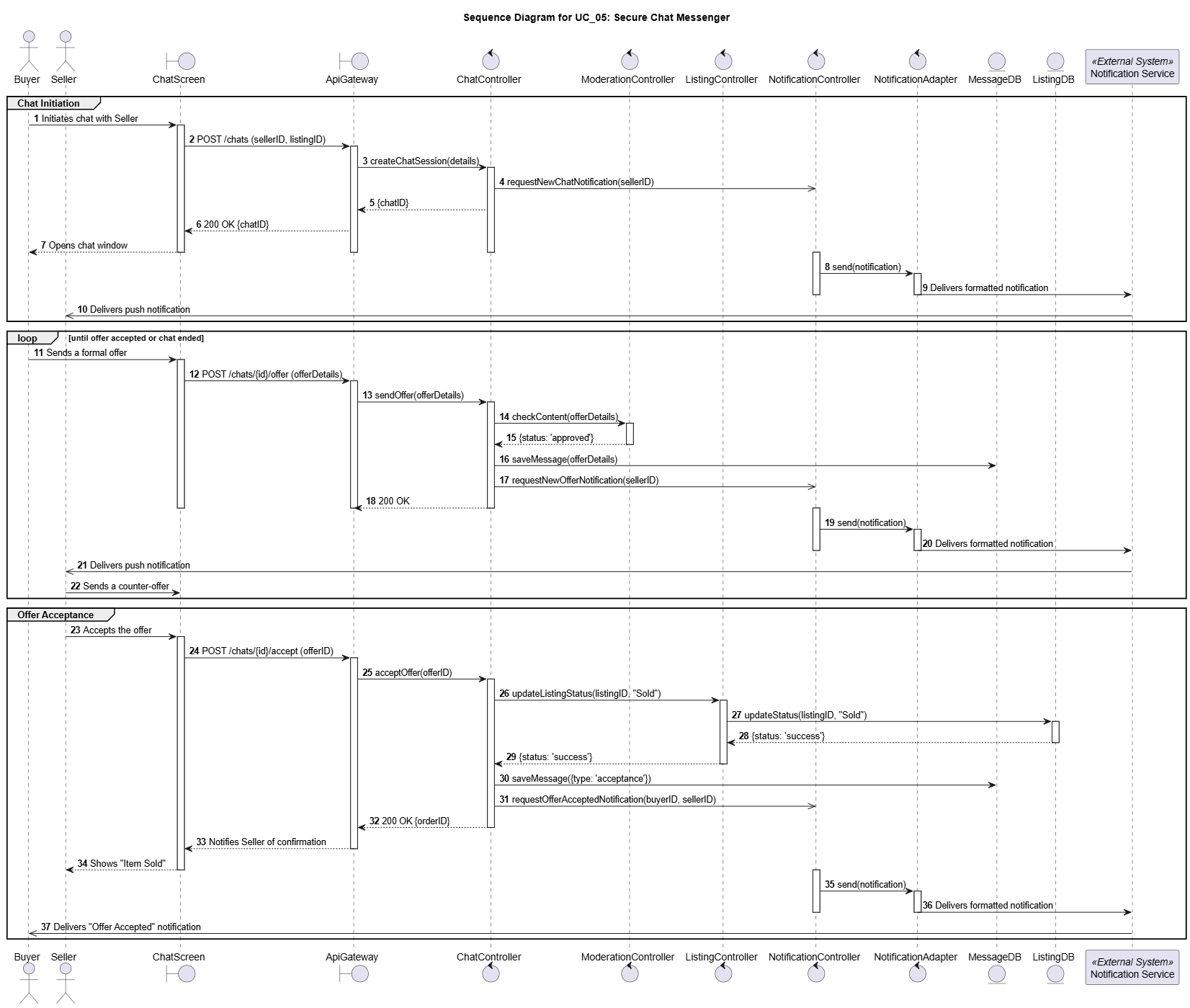


Figure 15: Sequence Diagram for UC05: Secure Chat Messenger

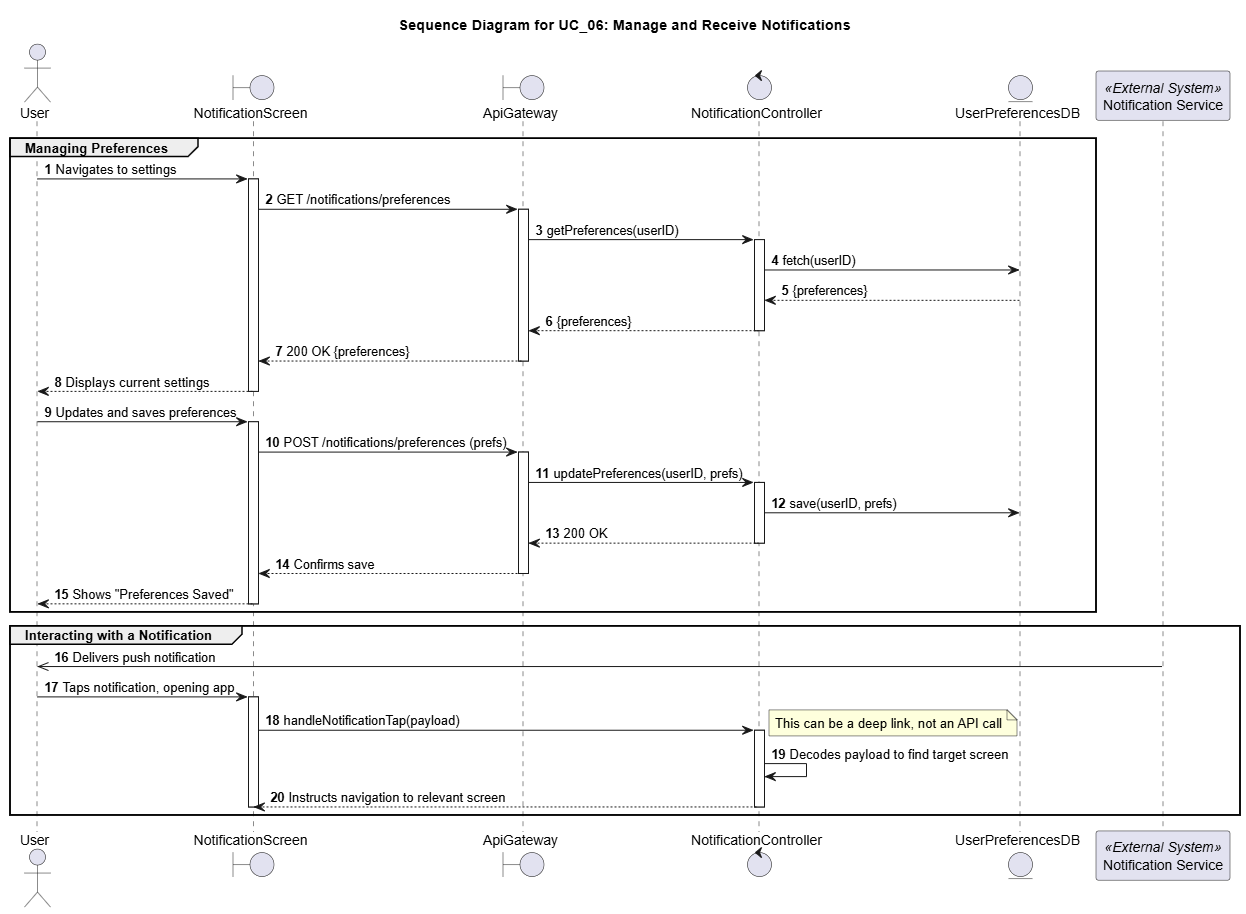


Figure 16: Sequence Diagram for UC06: Manage and Receive Notifications

1. Quality Scenarios

// A4. Quality Scenario Elicitation

// C4-1. Is there sufficient scenario elicitation affecting the architecture?

// C4-2. Is there sufficient review of the quality related to performance?

// C4-3. Is there sufficient review of the quality related to modifiability?

The following quality scenarios were identified through a combination of analyzing the core business drivers, reviewing common architectural concerns for C2C marketplaces, and conducting stakeholder interviews [representing Users (Buyers & Sellers), Business Stakeholders and Technical Stakeholders].

**B1. QAW (Quality Attribute Workshop)**

**B1.1. User Requirements**

* “Show me things near me or stuff that I could actually be interested in, not just random stuff.” *(Buyer)*
* "The search should understand what I'm *really* looking for, even if my search terms aren't perfect or exact." *(Buyer)*
* “Listing has to be super easy. Upload photos, add important details, and it should appear fast.” *(Seller)*

**B1.2. Business Leader Perspective:**

* "The platform must attract and retain users. Search and recommendations are key – they have to be better than competitors."
* ”We need to launch new features quickly to stay competitive. The architecture shouldn't slow us down."

**B1.3. Developer Perspective:**

* “Adding new features or fixing bugs shouldn't require rewriting huge parts of the system.”
* “It should be easy to add support for new listing categories or change recommendation algorithms without breaking everything."

**B1.4. Utility Tree**

|  |  |  |
| --- | --- | --- |
| **Classification** | **QS\_ID** | **Quality Scenario** |
| Performance | QS-01 | Search operation response time |
| Performance | QS-02 | Listing visibility time (new/updated) |
| Performance | QS-03 | Home Screen Loading time |
| Performance | QS-04 | Chat message delivery time |
| Performance | QS-05 | Listing Media Processing Time |
| Performance | QS-06 | Service auto-scaling based on load |
| Reliability | QS-07 | Monitoring of Cloud Microservices |
| Reliability | QS-08 | Chat messaging ordering & durability |
| Reliability | QS-09 | System recovery from errors/infrastructure failure |
| Maintainability | QS-10 | Flexibility to adopt new Recommendation algorithms |
| Maintainability | QS-11 | Flexibility to add new Category in item listing section |
| Maintainability | QS-12 | System operation cost-efficiency |
| Security | QS-13 | Personal Information & Off-platform Solicitation Protection (Chat) |
| Security | QS-14 | Listing Moderation & Malicious Content Prevention |
| Security | QS-15 | Prevent unauthorized access to MicroService |
| Security | QS-16 | Real-time flagging of high-risk sellers |
| Usability | QS-17 | Recommendation usefulness to end users |
| Usability | QS-18 | Search Result relevance |

**B2. Quality Scenarios Details**

|  |  |  |
| --- | --- | --- |
| **Category** | **Sub Category** | **Quality Scenario** |
| **Performance** | **Response Time** | **QS-01: Search operation response time** |
| When a user searches for an item, the system must return results within a very short time.  Measure: [Response Time] = [t\_result\_returned] – [t\_query\_received] at API gateway. |
| **Response Time** | **QS-02: Time taken for a new (or updates in existing) listing to be visible to the user** |
| When a seller publishes a new listing or changes an existing one, it should be discoverable by search and recommendation systems almost instantly Measure: [Latency] = [t\_visible\_in\_search] – [t\_published/t\_edited]. |
| **Response Time** | **QS-03: Home Screen Loading time** |
| When a user opens the homescreen, the time taken to load all the data should be as low as possible Measure: [Recommendation Latency] = [t\_results\_returned] – [t\_request\_initiated]. |
| **Response Time** | **QS-04: Chat message delivery time** |
| Messages between users must be delivered with minimal delay to enable smooth communication and negotiation flows. Measure: Delivery Latency = [t\_received] – [t\_sent]. |
|  | **QS-05: Listing Media Processing Time** |
| add some description here Measure: [Media Ready Latency] = [t\_preview\_available] – [t\_upload\_initiated] |
| **Scalability** | **QS-06: Service should be able to scale up/down as per the load on service** |
| System is able to adjust itself as per the traffic trend there by ensuring that it does not affect the end user. Measure: [Average throughput] >= N thousand request per sec |
| **Reliability** | **Fault Tolerance** | **QS-07: Monitoring of Cloud Microservices** |
| System should be able to monitor key service and applications. Alarms should be generated in case of any abnormality Measure: [Error percentage of key service metric (like 5xx error rates) should be almost 0) |
| **Durability** | **QS-08: Messaging ordering & durability** |
| Chats must never be lost and message ordering should be guaranteed, even with offline/reconnect. Measure: [Message Loss Rate] |
| **Availability** | **QS-09: System should be able to recover from any error in system or Server Infrastructure** |
| System should be able to handle any error in system or infrastructure failure intelligently without affecting end user Measure: [Time to recover] <= 'N' minutes |
| **Maintainability** | **Modifiability** | **QS-10: System should be flexible to adopt new Recommendation algorithms** |
| System should be adaptable to easily change to new algorithms for better recommendations and search features Measure: [Development Effort in months to integrate new Algorithms] |
| **Modifiability** | **QS-11: System should be flexible to add new Category in item listing section** |
| Addition of new category in items listing section should be done quickly without impacting the existing data. Measure: [Development Effort in weeks for implementing new category] |
| **Cost Efficiency** | **QS-12: System operation should be cost-efficient** |
| Ensures infrastructure costs are managed effectively, especially through auto-scaling. Measure: [Cost per Million requests] |
| **Security** | **Fraud Detection** | **QS-13: Personal Information & Off-platform Solicitation Protection** |
| Automatically detect and block sensitive information (phone numbers, emails) and off-platform payment solicitations (e.g., payment links) in chat to reduce fraud exposure and keep transactions secure. Measure: [Detection Accuracy] = correctly identified violations / total violations. [False Positive Rate] = incorrect blocks / total detections. |
| **Content Moderation** | **QS-14: Listing Moderation & Malicious Content Prevention** |
| Automatically prevent prohibited, unsafe, or malicious content at listing creation using text/image moderation, ensuring policy compliance and protecting buyers while minimizing false positives for legitimate sellers. Measure: [False Positive Rate] = incorrect blocks / total detections. |
| **Authentication** | **QS-15: System should prevent unauthorised access to MicroService** |
| System should implement mechanism to prevent unauthorised access to APIs Measure: [Number of Violations]/[total Count] |
| **Fraud Detection** | **QS-16: System should be able to flag high-risk sellers in real-time to prevent abuse** |
| High-risk sellers must be flagged early to prevent abuse. Listings from low-risk sellers should be promoted Measure: [Review Latency] |
| **Usability** | **Appropriateness** | **QS-17: Recommendation should be useful to end users** |
| Recommendation provided by our system should be beneficial for end user. In order to quantify this, we should have an A/B testing mechanism around recommended listings for actual user acceptance. Measure : [Click-Through-Rate] |
| **Appropriateness** | **QS-18: Search Result should be relevant** |
| System should display search results in order of relevance Recommended, most viewed etc. Measure : [Search Click-through-Rate] |

1. Quality Scenario Analysis

// A5. Quality Requirement Specification

// C5-1. Is quality scenario analysis appropriate? (Evidence)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Sub Category** | **Quality Scenario** | **Importance** | **Difficulty** | **Type** |
| **Performance** | **Response Time** | QS-01: Search operation response time should be low | H | H | NFR\_01 |
| **Response Time** | QS-02: Time taken for a new (or updates in existing) listing to be visible to the user should be low | H | M | QA\_05 |
| **Response Time** | QS-03: Home Screen Loading time should be low | H | M | NFR\_02 |
| **Response Time** | QS-04: Users should be able to get chat message quickly | M | M | QA\_09 |
| **Response Time** | QS-05: System should be able to process Media Content in new Listing quickly | M | M | QA\_08 |
| **Scalability** | QS-06: System should be able to scale the servers up/down as per the load on the service automatically | H | M | NFR\_03 |
| **Reliability** | **Fault Tolerance** | QS-07: System should have a mechanism of monitoring all the Micro services | M | M | QA\_10 |
| **Durability** | QS-08: System should ensure chat messaging ordering & durability | L | L | Ignore |
| **Availability** | QS-09: System should be able to recover from any error in system or Server Infrastructure | H | M | NFR\_04 |
| **Maintainability** | **Modifiability** | QS-10: System should be flexible to adopt new Recommendation algorithms | M | H | QA\_06 |
| **Modifiability** | QS-11: System should be flexible to add new Category in item listing section | L | L | Ignore |
| **Cost Efficiency** | QS-12: System should be Cost efficient | M | L | Ignore |
| **Security** | **Fraud Detection** | QS-13: System must block the sharing of contact information and any off the platform transaction. | H | M | QA\_03 |
| **Content Moderation** | QS-14: System should moderate all listings to prevent malicious or prohibited content from appearing on the platform | H | M | QA\_04 |
| **Authentication** | QS-15: System should prevent unauthorised access to MicroService | M | L | Ignore |
| **Fraud Detection** | QS-16: System should be able to flag high-risk sellers in real-time to prevent abuse | M | H | QA\_07 |
| **Usability** | **Appropriateness** | QS-17: Recommendation should be useful to users | H | M | QA\_02 |
| **Appropriateness** | QS-18: Search Result should be relevant | H | M | QA\_01 |

**QS-01: Search operation response time should be low**

This is the most critical performance metric as it directly supports the primary business driver of **Easy Discovery**. A slow search feature creates a frustrating user experience, undermining our goal of helping buyers find what they want quickly. Therefore, it is the **highest priority non-functional requirement** for the platform's success.

**QS-02: Time taken for a new (or updates in existing) listing to be visible to the user should be low**

This scenario is fundamental to the **Seamless Listing Experience**. Sellers expect their items to be discoverable immediately after publishing, and any delay breaks the "effortless" process and erodes seller trust. Minimizing this latency is a **high-priority architectural driver**.

**QS-03: Home Screen Loading time should be low**

The home screen is the main entry point for **Easy Discovery**, featuring personalized recommendations. A long loading time creates a poor first impression and can lead to user abandonment before they even begin browsing. Fast home screen performance is therefore a **non-negotiable, fundamental requirement** for the platform’s viability, hence making is **high-priority** NFR.

**QS-04: Users should be able to get chat message quickly**

Real-time communication is vital for negotiation and **Building Community Trust**. Delays in chat can make conversations feel disjointed and lead to frustration, but it follows the initial discovery and engagement phase. This makes ensuring low latency in chat an **important, medium-priority factor** for facilitating successful transactions.

**QS-05: System should be able to process Media Content in new Listing quickly**

This scenario is an important component of the **Seamless Listing Experience**. A slow or failing media upload process is a major friction point in the listing flow. Optimizing this is essential for a smooth seller journey, making it a **medium-priority quality attribute**.

**QS-06: System should be able to scale the servers up/down as per the load on the service automatically**

Scalability is a foundational requirement that underpins all business drivers. The system must maintain performance for all features during peak traffic, as a failure to scale would degrade the entire user experience. This is a **high-priority architectural concern** for long-term viability.

**QS-07: System should have a mechanism of monitoring all the Micro services**

Proactive monitoring is important for ensuring the platform remains reliable and trusted. However, it is an operational concern that supports the system rather than a direct, user-facing feature that drives the core architecture. It can be refined post-launch, which is why it is considered a **lower-priority quality attribute (QA\_11)**.

**QS-08: System should ensure chat messaging ordering & durability**

This scenario is critical for **Building Community Trust**. Lost or out-of-order messages during a negotiation can lead to significant confusion and broken deals, severely damaging user trust in the platform. Guaranteeing message integrity is therefore a **high-priority reliability requirement**.

**QS-09: System should be able to recover from any error in system or Server Infrastructure**

High availability is a fundamental pillar of a trustworthy platform. If the system cannot recover quickly from failures, users cannot discover items, list products, or communicate, eroding confidence across all business drivers. Therefore, automated and rapid system recovery is a **high-priority requirement**.

**QS-10: System should be flexible to adopt new Recommendation algorithms**

This attribute is key to the long-term success of the **Easy Discovery** driver. The ability to easily integrate new and improved recommendation models ensures the platform can adapt and improve the user experience over time. This makes it an **important, medium-priority attribute for future growth**.

**QS-11: System should be flexible to add new Category in item listing section**

While important for long-term business flexibility, adding new categories does not fundamentally influence the core real-time performance or reliability of the architecture. This functionality can be addressed with minor impact on the existing design. Therefore, **this scenario is considered a low-priority concern and can be dropped** as a primary architectural driver for now.

**QS-12: System should be Cost efficient**

Cost is a critical business metric, but optimizing it is a secondary operational goal compared to establishing core functionality and performance. The primary architecture must first ensure the system works reliably and scales effectively while cost can be tuned later. Therefore, **this scenario can be dropped** as a main architectural driver.

**QS-13: System must block the sharing of contact information and any off the platform transaction**

This is a cornerstone of **Building Community Trust**. The system must proactively protect users from fraud within chat to make the marketplace a safe environment for transactions. This is a **high-priority security requirement**.

**QS-14: System should moderate all listings to prevent malicious or prohibited content from appearing on the platform**

This scenario directly addresses the business driver of **Building Community Trust** by proactively detecting fraud. Automatically scanning new listings for policy violations is essential for protecting buyers and maintaining the platform's reputation, making this a **high-priority security feature**.

**QS-15: System should prevent unauthorized access to MicroService**

While preventing unauthorized inter-service communication is a fundamental security principle, its solution relies on standard industry patterns like API gateways and token-based authentication. This concern is addressed as a baseline implementation requirement rather than a unique architectural challenge that drives the overall design. Therefore, **this scenario is dropped as a primary architectural driver** from this analysis.

**QS-16: System should be able to flag high-risk sellers in real-time to prevent abuse**

This is a key preventative measure for **Building Community Trust** by flagging potential fraud before it impacts buyers. It is a complex feature that significantly enhances platform safety. It is therefore an **important, medium-priority architectural concern**.

**QS-17: Recommendation should be useful to end users**

This usability scenario is the ultimate measure of success for the **Easy Discovery** driver. A high click-through rate is direct evidence that the system is succeeding in helping users find items they want. This makes it a **vital, medium-priority metric** for guiding product development.

**QS-18: Search Result should be relevant**

This is the qualitative counterpart to search performance and is critical for **Easy Discovery**. A fast search that returns irrelevant results is useless. Ensuring the top results are highly relevant to the user's intent is **fundamental to the platform’s core value proposition**.

1. Candidate Architectures

// A6. Candidate Architecture Design

// C6-1. Are quality analysis and solution candidate appropriate?

// C6-2. Are performance analysis and solution candidate appropriate?

// C6-3. Are modifiability analysis and solution candidate appropriate?

In this section, we will explore Candidate Architectures targeting each Quality Attribute and Non-Functional Requirement we described in Section 2.2 and 2.3

**Targeted QA/NFR:**

* NFR\_01 (**Performance**): Search operation response time. This requires that the time between the system receiving a query and returning the results be less than or equal to 'N' milliseconds.
* QA\_01 (**Usability**): Search Result should be relevant. This measures the effectiveness of the search algorithm by quantifying user engagement with the results, such as the click-through rate.

1. Candidate Architecture Evaluation

// A7. Architecture Design

// C7-1. Is comparison analysis of colliding candidates appropriate? (evidence)

// C7-2. Is there sufficient complement of the selected candidate?

1. Final Architecture

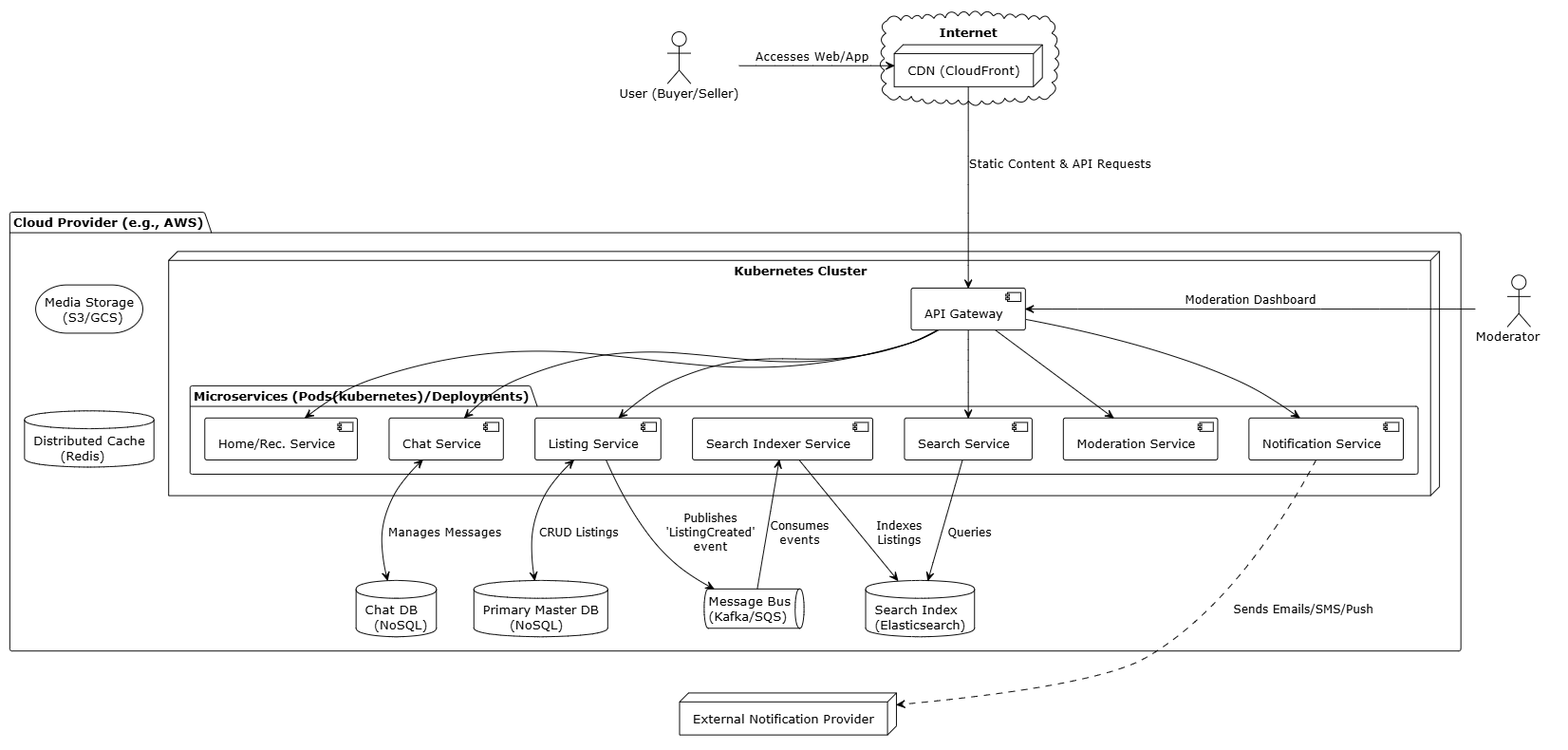
// A7. Architecture Design

// C7-3. Is there right integration into the final architecture?

// C7-4. Is there appropriate risk management of the final architecture?

**F1.1 Deployment View**

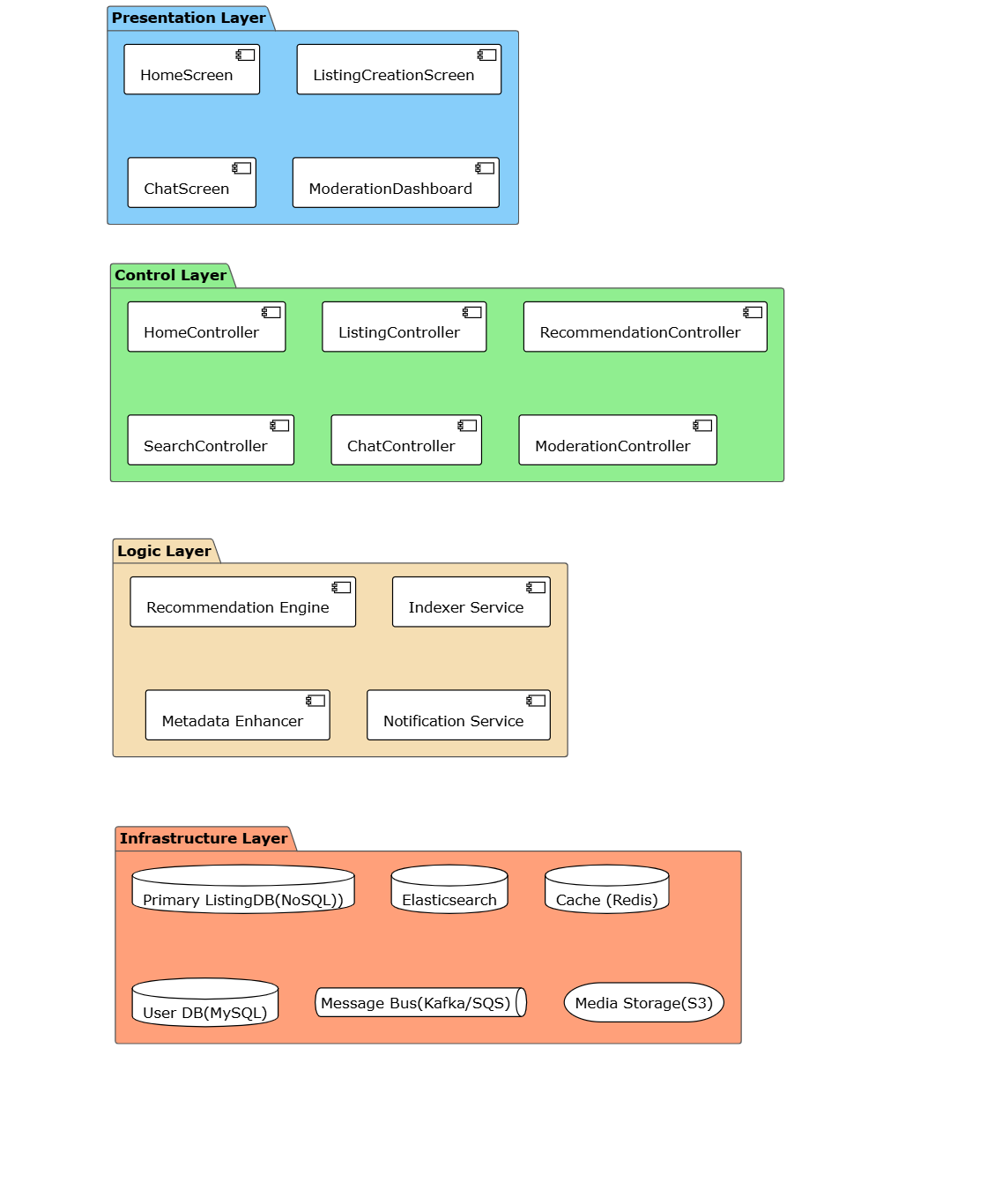
**Overview.** The marketplace is deployed as a cloud-native, Kubernetes-orchestrated system with an edge tier (DNS/CDN/WAF/API-Gateway), a services tier (microservices in K8s), and managed data plane (RDS/PSQL, Redis, Kafka/SNS+SQS, S3/GCS, OpenSearch/ES, Warehouse). Services scale horizontally and communicate synchronously via REST/WebSockets and asynchronously via an event bus for decoupling and resilience.



**Figure: Deployment View for C2C MarketPlace**

**F1.2 Module View**

The system's modules are organized into a four-layer architecture to ensure a clear separation of concerns and promote modifiability. The **Presentation Layer** contains all UI components, while the **Control Layer** handles incoming user requests. The **Logic Layer** executes core business processes, and the **Infrastructure Layer** manages data persistence and external services.



**Figure: Module View for C2C MarketPlace**

**F1.3 Risks**

**1. Data Inconsistency due to Asynchronous Processing**

* **Risk**: Our extensive use of an **event-driven architecture** for decoupling services introduces a primary risk of data inconsistency. A failure in the asynchronous pipeline could lead to the search index becoming stale (e.g., a sold item still appears in search results), product images failing to process after upload, or cached data becoming outdated.
* **Mitigation Plan**: This risk will be managed by implementing a **Dead-Letter Queue (DLQ)** for the message bus. If a service fails to process an event after several retries, the message will be moved to the DLQ, triggering an immediate alert for investigation without losing the update. For caching, we will supplement this with an event-driven cache invalidation strategy and short Time-to-Live (TTL) policies.

**2. Cascading Failures in Microservice Dependencies**

* **Risk**: In a distributed system, a single slow or failing downstream service can block upstream callers, leading to resource exhaustion and a chain reaction of failures that can bring down the entire platform.
* **Mitigation Plan**: We will implement the **Circuit Breaker pattern** to isolate failures. The primary risk of this solution is misconfiguration (thresholds being too sensitive or too lenient). This will be mitigated by using dynamic, adaptive thresholds based on real-time service performance and by making the state of all circuit breakers a key metric in our monitoring dashboards.

**3. Algorithmic Bias Creating an Unfair Marketplace**

* **Risk**: The decision to use **Static Business-Rule Re-ranking** to boost listings from highly-rated sellers creates a significant risk of algorithmic bias. This can lead to a "rich-get-richer" feedback loop where new sellers are perpetually buried, stagnating inventory diversity and harming the platform's long-term health.
* **Mitigation Plan**: To counteract this bias, the re-ranking logic will include rules that provide temporary **"new seller" or "first listing" boosts**. This gives new participants a fair chance at gaining initial visibility. We will also actively monitor marketplace fairness metrics to ensure the ecosystem remains healthy and competitive.

**4. Real-time Chat Unreliability and Disconnections**

* **Risk**: Using a **Pub/Sub pattern with persistent WebSocket connections** introduces the risk of managing a large number of stateful connections. If a server node handling these connections fails, all connected users will be abruptly disconnected, leading to a poor user experience and potentially lost messages.
* **Mitigation Plan**: This will be mitigated by building a **resilient WebSocket gateway layer** coupled with robust client-side reconnection logic. Client applications will be designed to automatically re-establish their connection to a new healthy node and resynchronize their chat history, ensuring the experience is seamless and no data is lost.

**5. High Latency in Chat due to Synchronous Moderation**

* **Risk**: The decision to use a **Synchronous, Real-time Moderation Gateway** for chat messages introduces a critical risk of adding latency to every message sent. This could make the chat experience feel sluggish and unresponsive, violating a key quality attribute for real-time communication (QA\_09).
* **Mitigation Plan**: We will mitigate this by engineering the ModerationService for extremely low latency. This will be achieved using a **tiered-check system**, where fast, cached keyword and regex checks handle most cases, and only ambiguous messages are passed to slower, more complex machine learning models.

**6. High Operational Costs from Advanced AI Features**

* **Risk**: Implementing **Listing Metadata Enhancement using an LLM** introduces a significant and potentially unpredictable operational cost due to the per-API call pricing of large language models. A secondary risk is the generation of inaccurate or irrelevant tags that could pollute search results.
* **Mitigation Plan**: To manage costs, a **tiered enrichment strategy** will be implemented. A smaller, cheaper model will handle initial analysis, and only high-value or ambiguous listings will be escalated to the more expensive LLM. To ensure quality, generated tags will be validated against a confidence score and their impact will be measured via A/B testing.

1. Architecture Evaluation(ATAM)

// A10. Architecture Evaluation

// C10-1. Are there sufficient quality scenarios evaluating architecture?

// C10-2. Are there sufficient architectural decisions identified?

// C10-3. Is the analysis of design decisions appropriate? (evidence)

// C10-4. Are the mitigation plans to the risk factors appropriate?