TrueBite

AI-Enabled Restaurant Order & Delivery System Software Requirements Specification (SRS) For Core System (Ordering, Delivery, Finance, Reputation, and AI Chatbot)

Version <2.0>
Date: November 17, 2025
Project Team: Arsenii Chan, Angus Chen, Nick Kontonicolaou, Diana Lucero, and Bek Shiribaiev

[Note: The following template is provided for use with the Rational Unified Process. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

Revision History

Date	Version	Description	Author
20/Oct/2025	1.0	Initial SRS Document; defined system purpose and scope, outlined use-case model, specified actors, and detailed specific and supplementary requirements	Diana Lucero, Arsenii Chan, Angus Chen, Nick Kontonicolaou, Bek Shiribaiev
23/Oct/25	1.1	Refined use-case reports, added non-functional requirements and appendices	Team TrueBite
10/Nov/2025	1.5	Phase II Initial Development: - Expanded use-case descriptions - Added sequence diagrams - Developed Petri net models	
12/Nov/2025	1.8	Technical Documentation: - Added pseudo-code for UC-01, 02 - Developed database schema - Documented API endpoints	
15/Nov/202	1.9	Diagram Completion: - Completed all sequence diagrams (Diagrams) - Finalized 3 Petri net models - Created use-case survey diagram - Designed ER diagram	
17/Nov/202	2.0	Phase II Final Submission: - Integrated all diagrams - Completed all 15 use cases - Added GUI mockups - Finalized supplementary reqs - Added LLM integration details - Comprehensive quality review	

Table of Contents

Software Requirements Specification

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document provides a detailed description of the requirements for the TrueBite AI-Enabled Restaurant Order and Delivery System. It defines both functional and non-functional requirements for the development team, stakeholders, and testing personnel.

The system leverages Large Language Model (LLM) technology to provide intelligent, AI-driven customer service and integrates a local knowledge base for restaurant-specific information.

TrueBite supports three user categories:

- Employees – chefs, delivery personnel, and a manager
- Customers – registered users and VIP members
- Visitors – users who can browse menus and interact with the AI chatbot

The platform manages ordering, delivery bidding, finance handling, reputation management, and AI-based customer service, ensuring a seamless experience for all parties.

1.1 Scope

TrueBite is a **full-stack web application** developed with:

- **Backend:** Flask (Python)
- **Frontend:** React.js
- **Database:** PostgreSQL

Core Features:

- Menu browsing and food ordering with image-based GUI
- Personalized dashboards for users based on history
- AI-powered chatbot integrated with a local knowledge base + LLM fallback
- Reputation management for chefs, delivery people, and customers
- Financial management with deposits, balances, and automatic warnings
- Human resource logic for employee promotion/demotion
- Delivery bidding system with manager assignment
- A creative AI feature (e.g., predictive ordering, AR menu, or image-based search)

The system facilitates all aspects of restaurant operation: **menu management**, **order processing**, **delivery coordination**, and **customer engagement**, serving as an intelligent all-in-one management platform.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition

LLM	Large Language Model – AI model for natural language understanding
VIP	Very Important Person – premium customer tier unlocked by ≥$100 spent or ≥3 successful orders
GUI	Graphical User Interface – visual front-end of the system
RAG	Retrieval-Augmented Generation – technique integrating a local knowledge base with an LLM
DBMS	Database Management System – PostgreSQL used for relational data storage
JWT	JSON Web Token – standard for user authentication
Registered Customer	User with an account who can place and rate orders
Visitor	User who can browse menus but cannot order
Knowledge Base	Local database of restaurant FAQs, dishes, and AI-learned information
Reputation Score	Metric determined by complaints, compliments, and ratings
Warning	Penalty issued for violations such as unpaid orders or frivolous complaints
Blacklist	List of banned users who cannot re-register

1.4 References

1. IEEE Std 830-1998 – *IEEE Recommended Practice for Software Requirements Specifications*
2. Ollama Documentation – https://ollama.ai
3. Hugging Face Model Hub – https://huggingface.co/models
4. PostgreSQL Documentation – https://www.postgresql.org/docs/
5. Flask and React Official Documentation
6. CSC 322 Course Project Description (Prof. Jie Wei, Fall 2025)

1.5 Overview

This document is organized into four main sections:

- **Section 1 – Introduction:** Provides purpose, scope, and references.
- **Section 2 – Overall Description:** Describes the system's context, architecture, and user roles.

- **Section 3 – Specific Requirements:** Lists functional use cases and non-functional (supplementary) requirements.

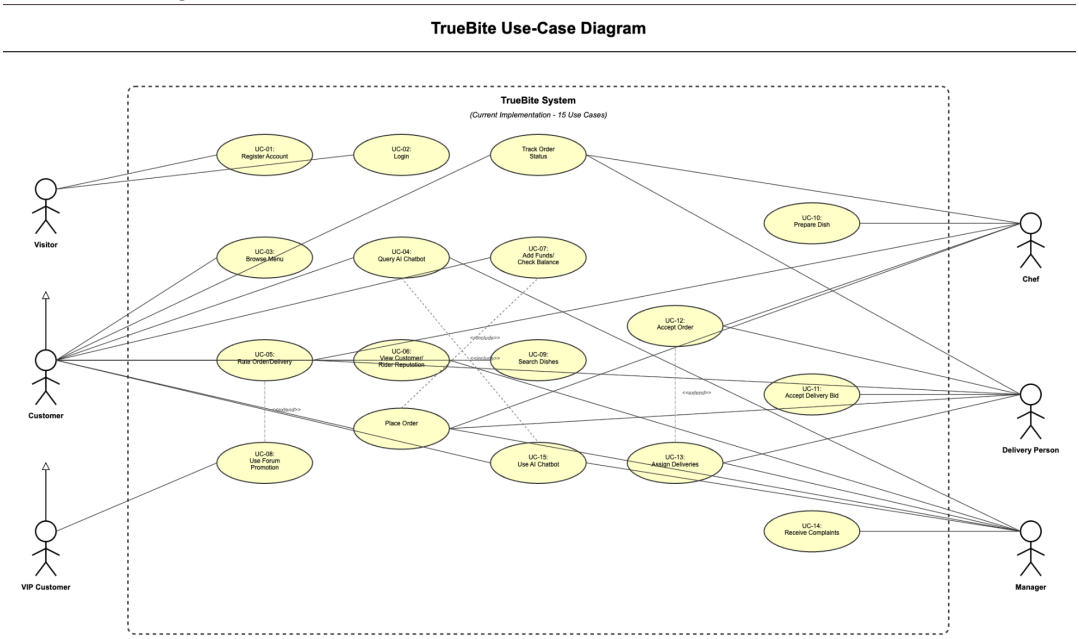
- **Section 4 – Supporting Information:** Includes diagrams, mockups, and appendices.

The document outlines how the system integrates multiple modules — including ordering, AI chatbot, and human resource management — and how they interact through defined APIs and database schemas

2. Overall Description

2.1 Use-Case Model Survey

2.1.1 Use-Case Diagram



This use-case diagram illustrates TrueBite's functional requirements across six actors: Visitor, Customer, VIP Customer, Chef, Delivery Person, and Manager.

Key Features:

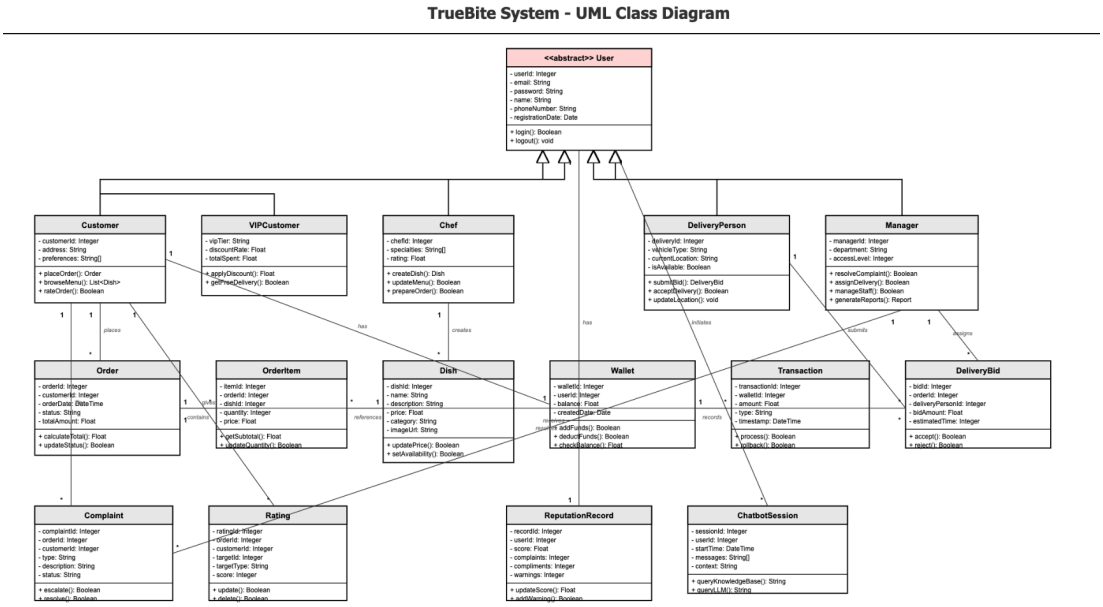
- **Inheritance**: VIP Customer inherits all Customer capabilities plus exclusive benefits (5% discount, free deliveries, special dishes)
- **Shared Use Cases**: Multiple actors share UC-02 (Log In) and UC-04 (Query AI Chatbot), demonstrating cross-actor interactions
- **Bidirectional Ratings**: UC-05/UC-06 enable customers to rate service providers and vice versa
- **System Boundary**: The dashed box contains 15 implemented use cases in the current phase

Actor Capabilities:

- **Visitor**: Browse, register, and query AI chatbot
- **Customer**: Full ordering, payment, rating, and forum access
- **VIP Customer**: Enhanced customer benefits with automatic discounts
- **Chef**: Kitchen operations including menu management and dish preparation
- **Delivery Person**: Competitive bidding and order fulfillment

- **Manager**: System oversight, complaint resolution, and delivery assignment

2.1.2 Class Diagram

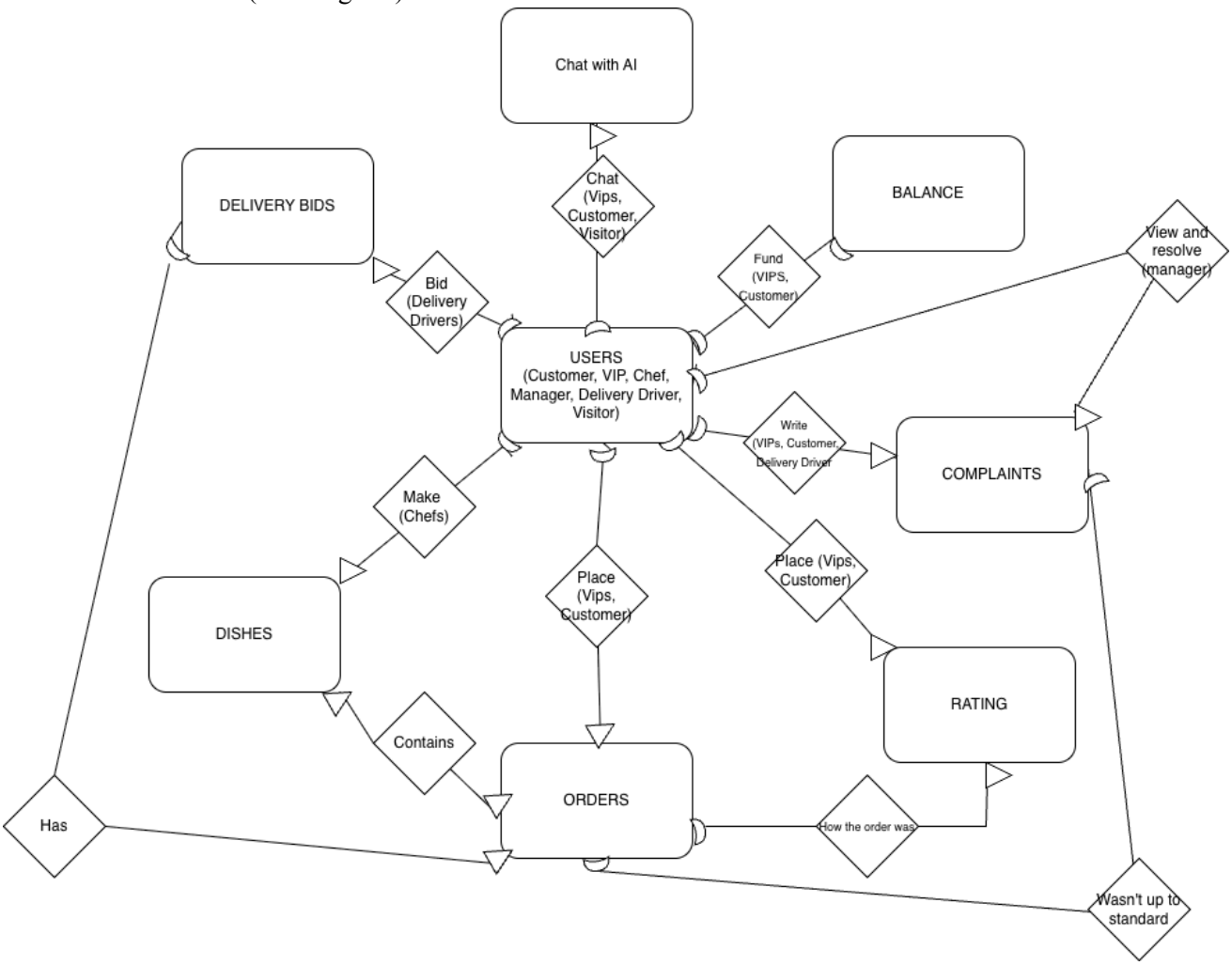


This UML class diagram shows TrueBite's object-oriented structure with five user types (Customer, VIPCustomer, Chef, DeliveryPerson, Manager) inheriting from an abstract User class. Key entities include Order, Dish, Wallet, Transaction, Complaint, Rating, and DeliveryBid.

The diagram illustrates relationships between classes: customers place orders containing order items that reference dishes; chefs create and manage dishes; delivery personnel submit bids for orders; and the manager oversees complaint resolution and delivery assignment. The ReputationRecord class aggregates ratings, complaints, and compliments for performance tracking.

This design provides the data structures and business logic supporting all system use cases documented in Section 3.1.2.

2.1.3 Data Model (ER Diagram)



This ER diagram shows TrueBite's database schema implemented in PostgreSQL. The USERS entity serves as the central table containing all actor types (Customer, VIP, Chef, Manager, Delivery Driver, Visitor) with a user_type discriminator.

Key Relationships:

Order Flow: USERS place ORDERS, which contain multiple DISHES, and are fulfilled through DELIVERY BIDS

Reputation: USERS write COMPLAINTS and RATING records for quality control

Finance: USERS maintain BALANCE accounts for transaction management

AI Support: All USERS can Chat with AI for system assistance

Cardinalities:

One-to-Many: USER → ORDERS, DISH → DELIVERY BIDS, USER → COMPLAINTS

Many-to-Many: ORDERS ↔ DISHES (via order items), USERS ↔ Chat sessions

One-to-One: USER → BALANCE

2.2 Assumptions and Dependencies

Assumptions:

- Users access the system via modern web browsers.
- Ollama or Hugging Face API access is available for AI queries.
- The PostgreSQL database server is active and accessible.
- JWT-based authentication ensures session security.
- The creative feature depends on open-source libraries (e.g., YOLO/DINO, speech recognition APIs).

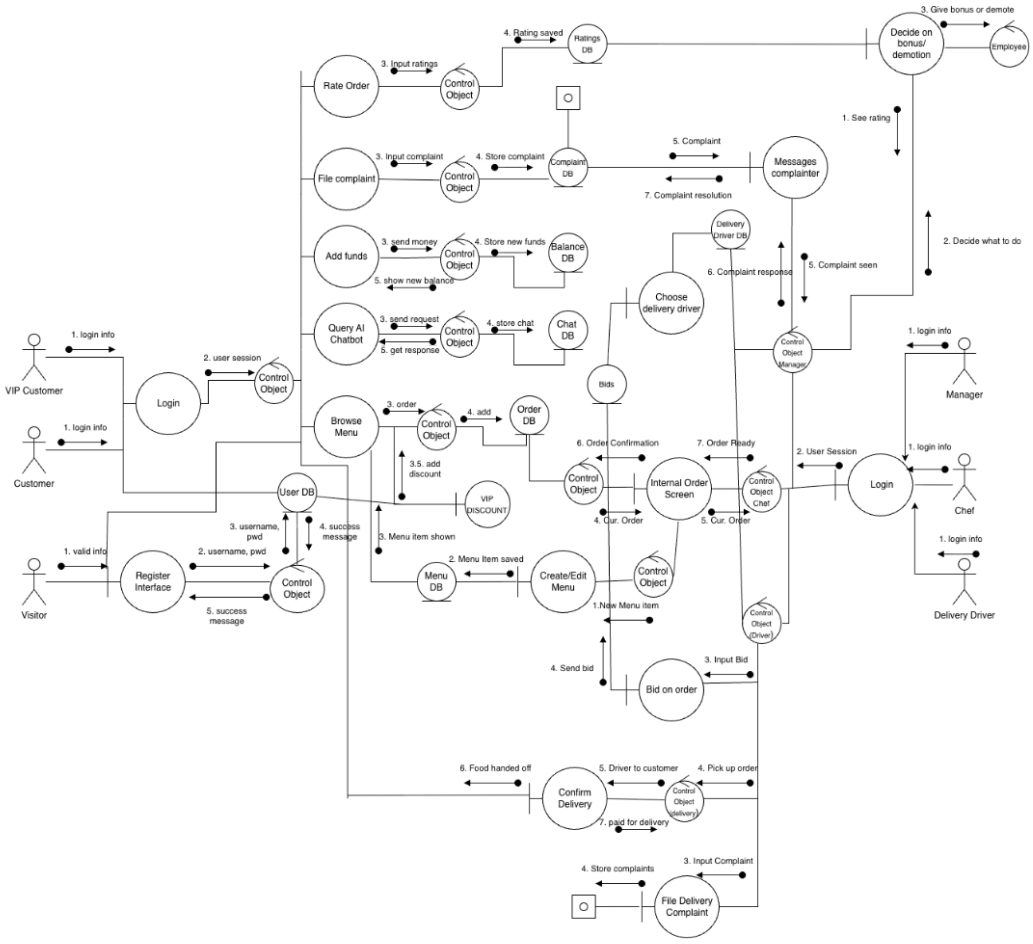
Primary Functions:

1. **User Management:** Registration, authentication, VIP upgrades, account status.
2. **Menu & Ordering:** Visual dish catalog, cart, personalized suggestions, order tracking.
3. **AI Chatbot:** Knowledge base lookup → fallback to LLM; user feedback integration.
4. **Reputation System:** Ratings, complaints, compliments, and warnings.
5. **Finance Management:** Deposits, payments, balance verification, and transaction history.
6. **Human Resources:** Performance tracking, salary adjustment, and delivery bidding.
7. **Discussion Forums:** Topics on chefs, dishes, delivery performance, and moderation.

3. Specific Requirements

3.1 Use-Case Reports

3.1.1 System Collaboration Overview



This collaboration diagram illustrates the interaction pathways between TrueBite's six actors and the system's control objects and entities. The numbered sequences show operational flows across different scenarios.

Primary Flows:

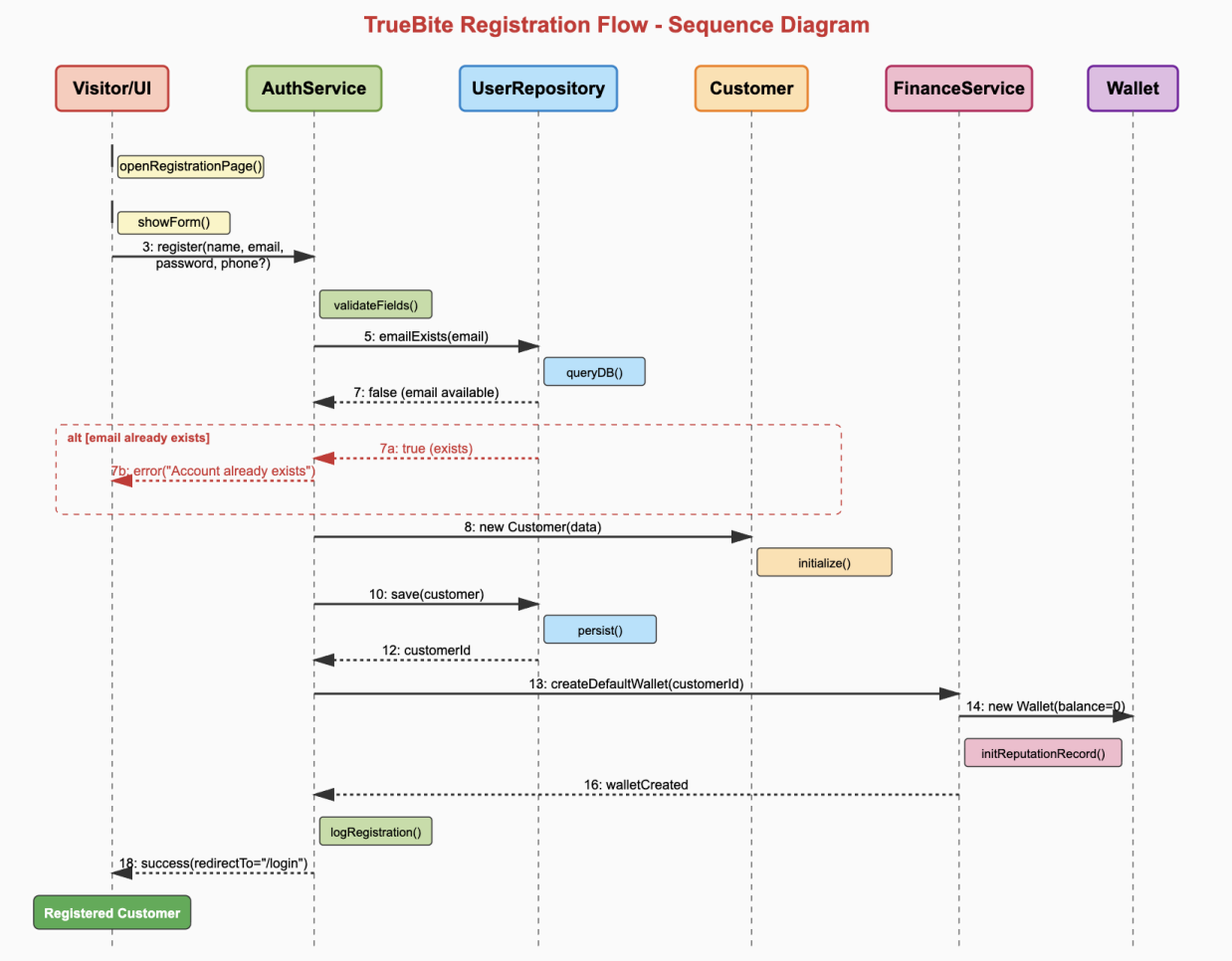
1. **Authentication:** All actors authenticate through Login control object (UC-02)
2. **Order Processing:** Customer → Menu → Rate Order → Add Funds → Chef → Delivery (UC-03, 04, 05, 10, 11, 12)
3. **Reputation Management:** Ratings and complaints flow through File complaint control with Manager oversight (UC-06, 07, 14)
4. **Delivery Bidding:** DeliveryPerson bids → Manager assigns → Delivery execution (UC-11, 13, 12)
5. **AI Support:** Query AI Chatbot accessible to all users (UC-15)
6. **Financial Operations:** Add funds and Wallet/Balance entities manage transactions (UC-05)

Key Observations:

- **Cross-actor interactions:** Both Customers and Delivery Personnel can file complaints, demonstrating interaction between "normal users and insiders"
- **Service coordination:** Complex operations involve multiple control objects (e.g., order placement uses Menu, Rate Order, Add Funds, and Chef controls)
- **Manager oversight:** Manager supervises complaints, delivery assignment, and employee performance across the system

- **Universal services:** Login and AI Chatbot are accessible to all actor types

3.1.2 Use-Case Reports



UC-01: Register Account

Actor: Visitor
Goal: Become a registered customer.

Normal Scenario

1. Visitor opens registration page.
2. System shows registration form (name, email, password, optional phone).
3. Visitor fills all required fields and submits.
4. `AuthService` validates fields and checks if email is unique.
5. On success, system creates a `Customer` record and associated `Wallet` with 0 balance and default `ReputationRecord`.

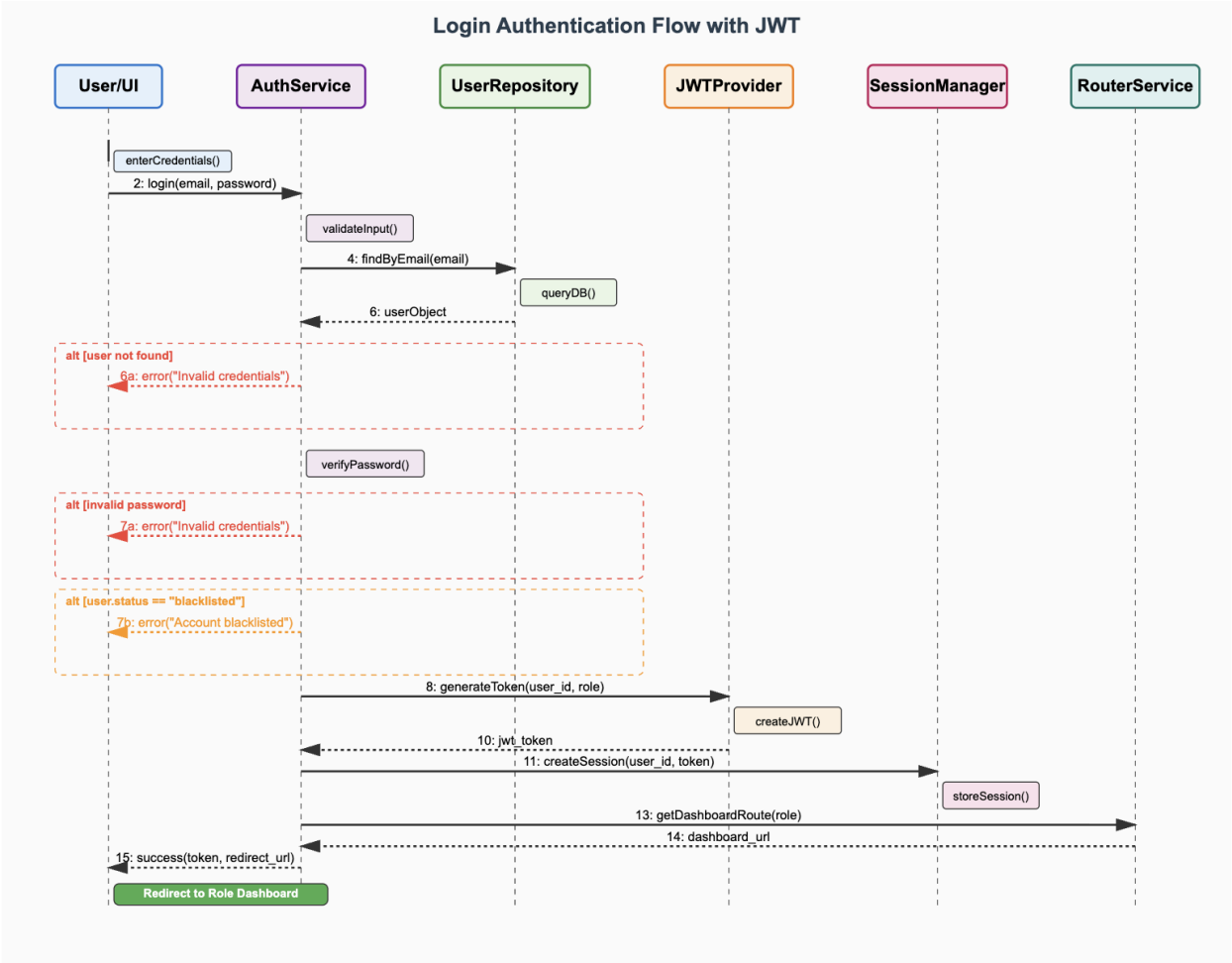
6. System logs the registration and redirects the user to login or dashboard.

Exceptional Scenarios

- **E1 – Missing/invalid fields**
    - If any required field is missing or invalid, system shows a validation error and does not create the account.
- **E2 – Email already in use**
    - If an email exists, the system shows “Account already exists” and asks user to log in instead.

Collaboration/Sequence Diagram (text description)

- Objects: `Visitor/UI`, `AuthService`, `Customer`, `Wallet`, `UserRepository`.
- Messages:
    - `UI → AuthService: register(formData)`
    - `AuthService → UserRepository: emailExists(email?)`
    - `AuthService → UserRepository: save(Customer)`
    - `AuthService → FinanceService: createDefaultWallet(customerId)`



UC-02: Log In

Actor: Registered Customer / VIP / Chef / DeliveryPerson / Manager

Normal Scenario

1. User enters email and password on login page.
2. `AuthService` validates credentials, checks password hash.
3. On success, system issues JWT/session token and loads role-specific dashboard.

Exceptional Scenarios

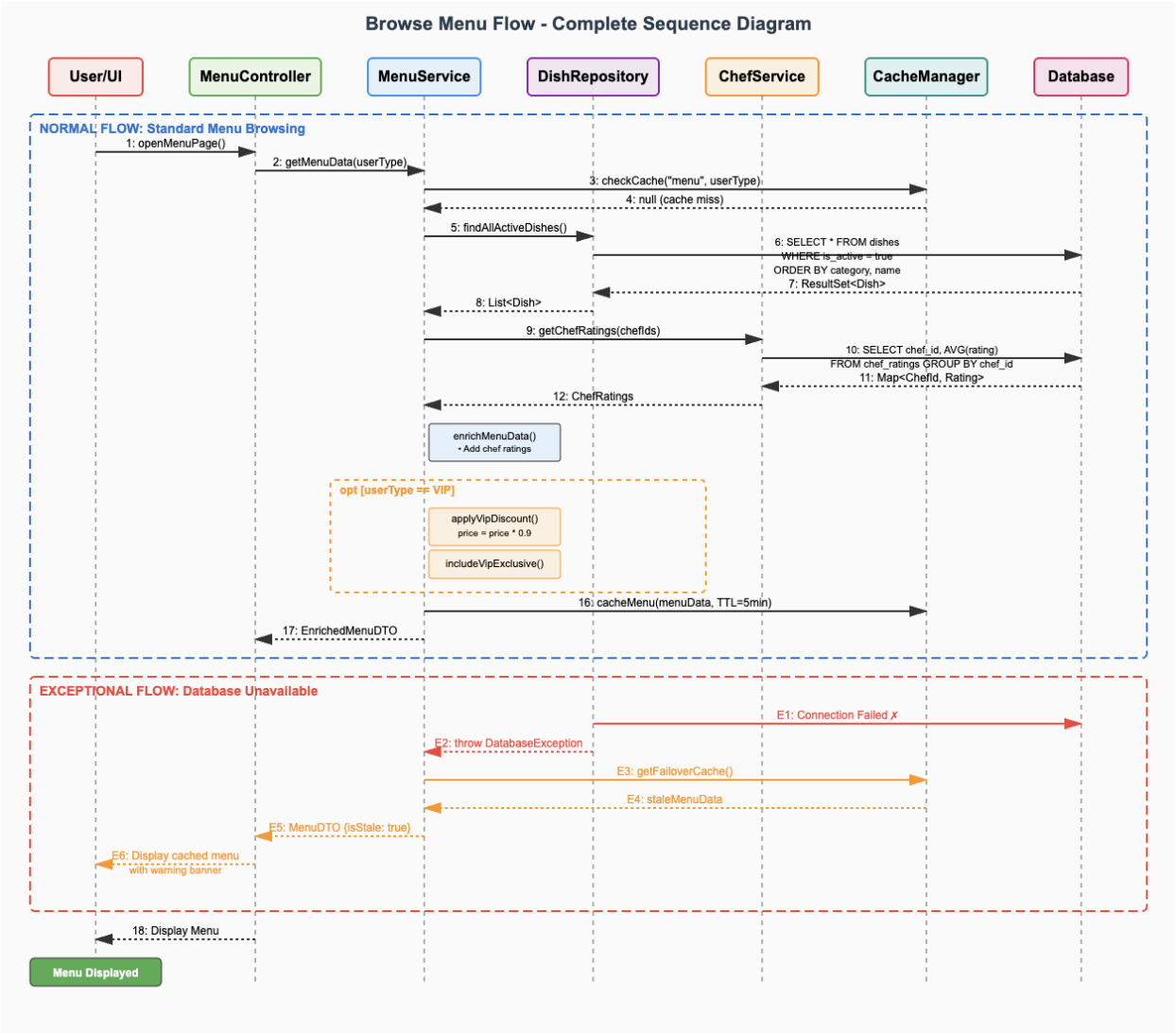
Invalid credentials → error message, no session created.

Locked/blacklisted user → access denied, show contact message.

Collaboration/Sequence Diagram

- `UI → AuthService: login(email, password)`

- `AuthService → UserRepository: findByEmail()`
- `AuthService: verifyPassword()`
- `AuthService → JWTProvider: generateToken()`



UC-03: Browse Menu

Actor: Visitor / Customer / VIP

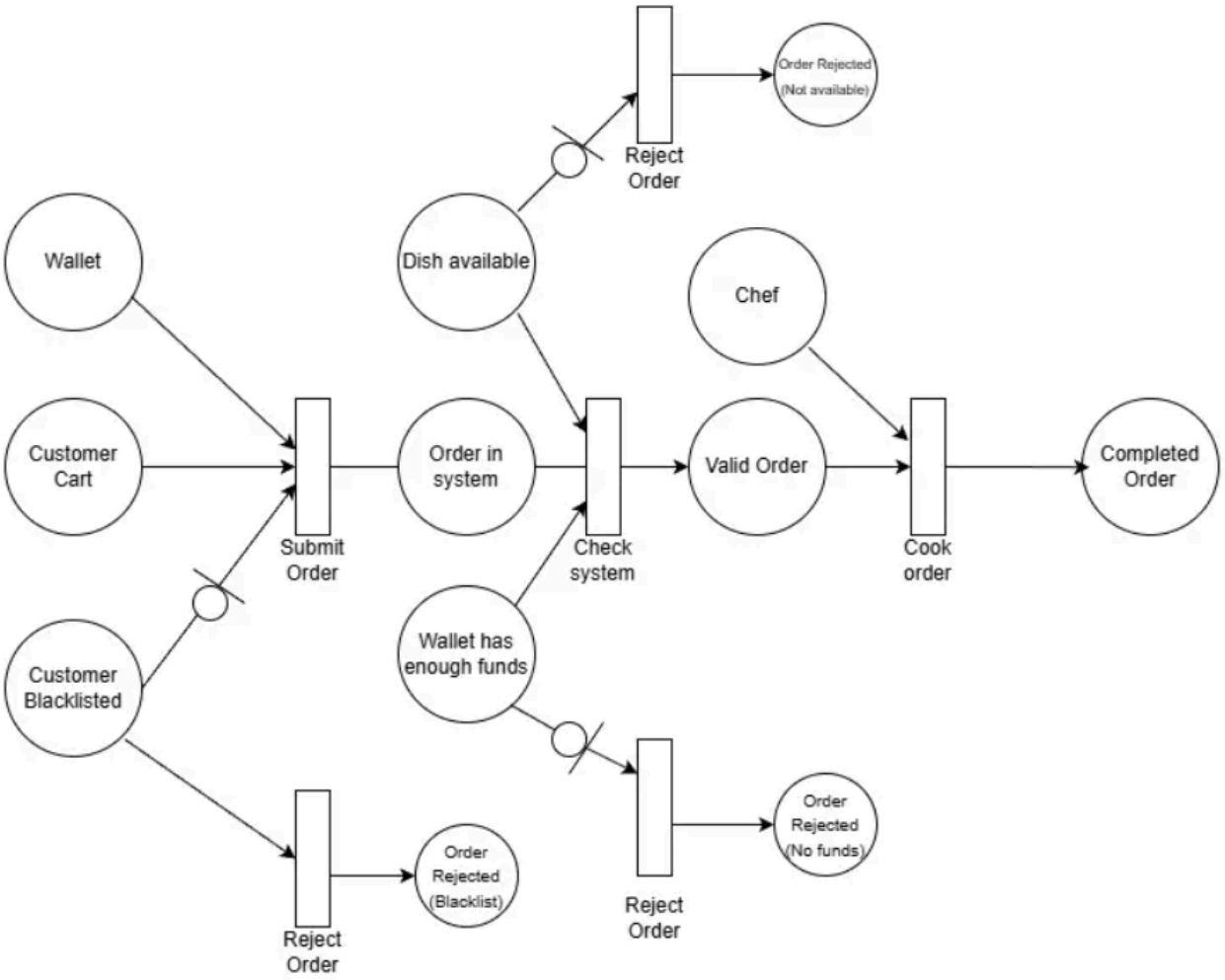
Normal Scenario

1. User opens Menu page.
2. `MenuService` fetches all active `Dish` records (with availability, price, tags).
3. System displays dishes, categories, and chef ratings.

Exceptional Scenario

- DB unavailable: show “Service temporarily unavailable”.

Diagram: Simple sequence: `UI → MenuService → DishRepository`.



UC-04: Place Order (Petri net UC #1)

Actor: Customer / VIP

Normal Scenario

1. User selects dishes and quantities into a cart.
2. User clicks “Place Order”.
3. `OrderService` validates cart (dish existence, availability).
4. `FinanceService` checks `Wallet` balance.

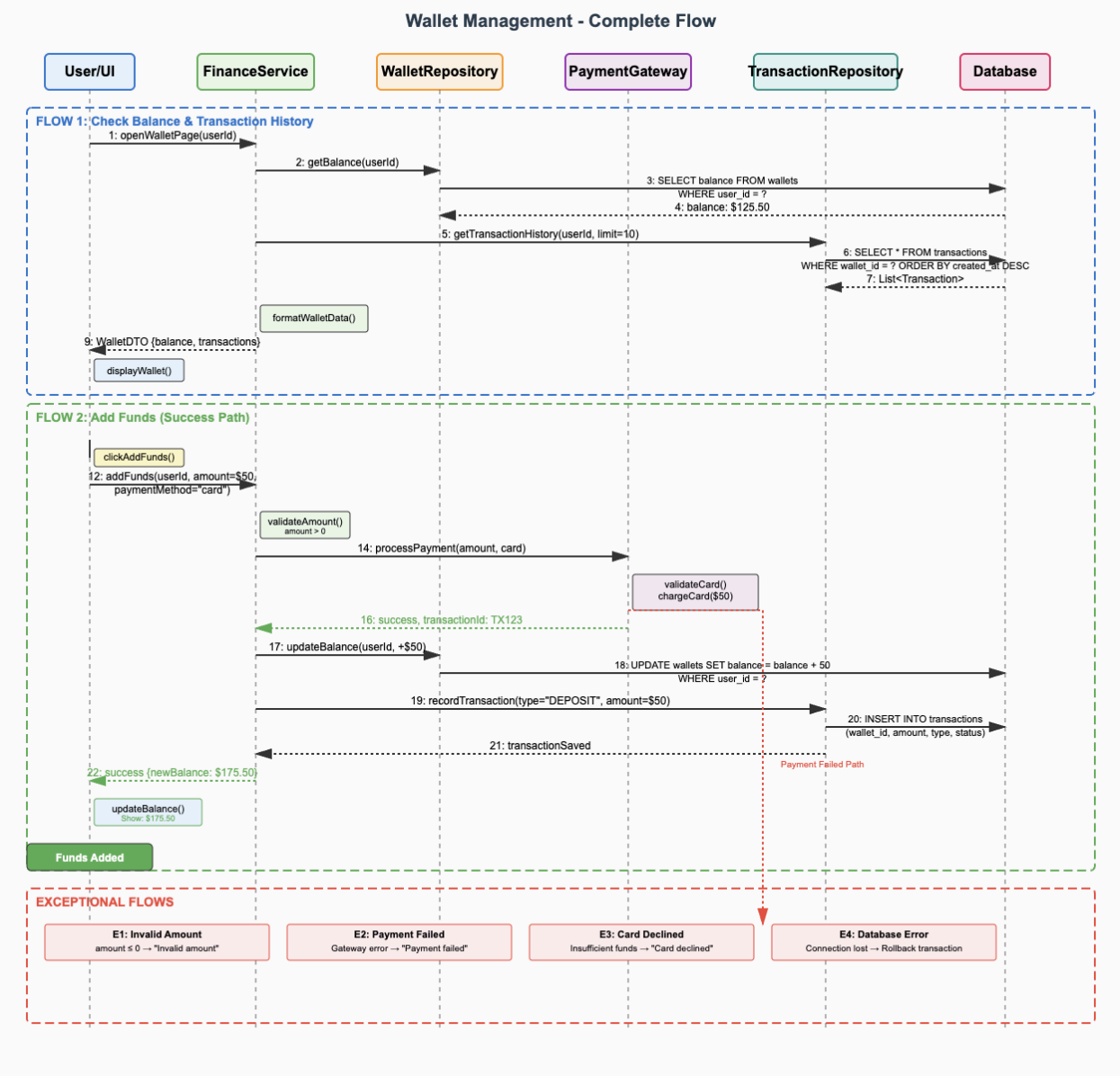
5. If sufficient, `FinanceService` deducts total price and records a `Transaction`.
6. `OrderService` creates `Order` + `OrderItem`s with status = `CREATED`.
7. `OrderService` assigns order to a `Chef` queue (`IN_KITCHEN`) and notifies kitchen.
8. System shows order confirmation and tracking page.

Exceptional Scenarios

- **E1 – Insufficient balance**
    - System rejects order, suggests user to add funds.
- **E2 – Dish unavailable**
    - System refuses order or partially adjusts cart.
- **E3 – Customer is blacklisted**
    - System blocks ordering, displays “Account restricted”.

Petri Net Description (you’ll draw it)

- **Places:**
    - `P_NewCart` – built, not submitted
    - `P_ValidOrder` – order validated & paid
    - `P_InsufficientFunds`
    - `P_BlacklistedCustomer`
    - `P_OrdersInKitchen`
    - `P_OrdersRejected`
- **Transitions:**
    - `T_SubmitOrder` – from `P_NewCart` to either `P_ValidOrder` / `P_InsufficientFunds` / `P_BlacklistedCustomer`
    - `T_SendToKitchen` – from `P_ValidOrder` to `P_OrdersInKitchen`
    - `T_RejectOrder` – to `P_OrdersRejected`
- **Tokens:**
    - Each token in `P_NewCart` represents a pending cart; firing transitions move it through the process.



UC-05: Add Funds / Check Balance

Actor: Customer / VIP

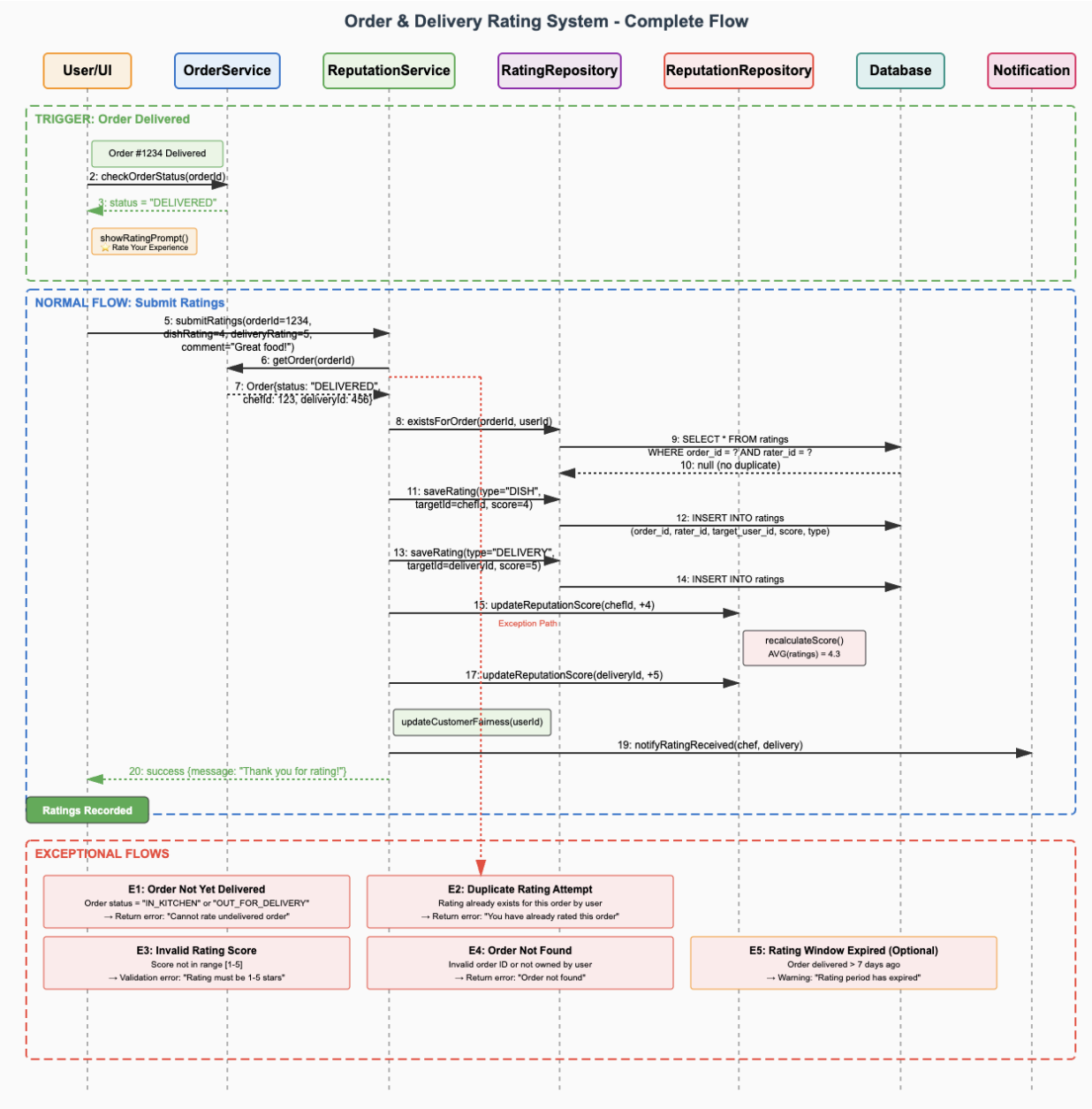
Normal Scenario

1. User opens Wallet page.
2. System shows current balance and transaction history (read-only).
3. User clicks “Add Funds”, enters amount and payment info.
4. `FinanceService` validates amount and processes payment (simulated).
5. On success, `Wallet.balance` increases; `Transaction` record created.
6. System confirms updated balance.

Exceptional Scenarios

- Invalid payment info / external payment failure → no balance update; show error.
- Negative or zero amount → validation error.

Diagram: Sequence: `UI → FinanceService → PaymentGateway (mock) → WalletRepository`.



UC-06: Rate Order / Delivery

Actor: Customer / VIP

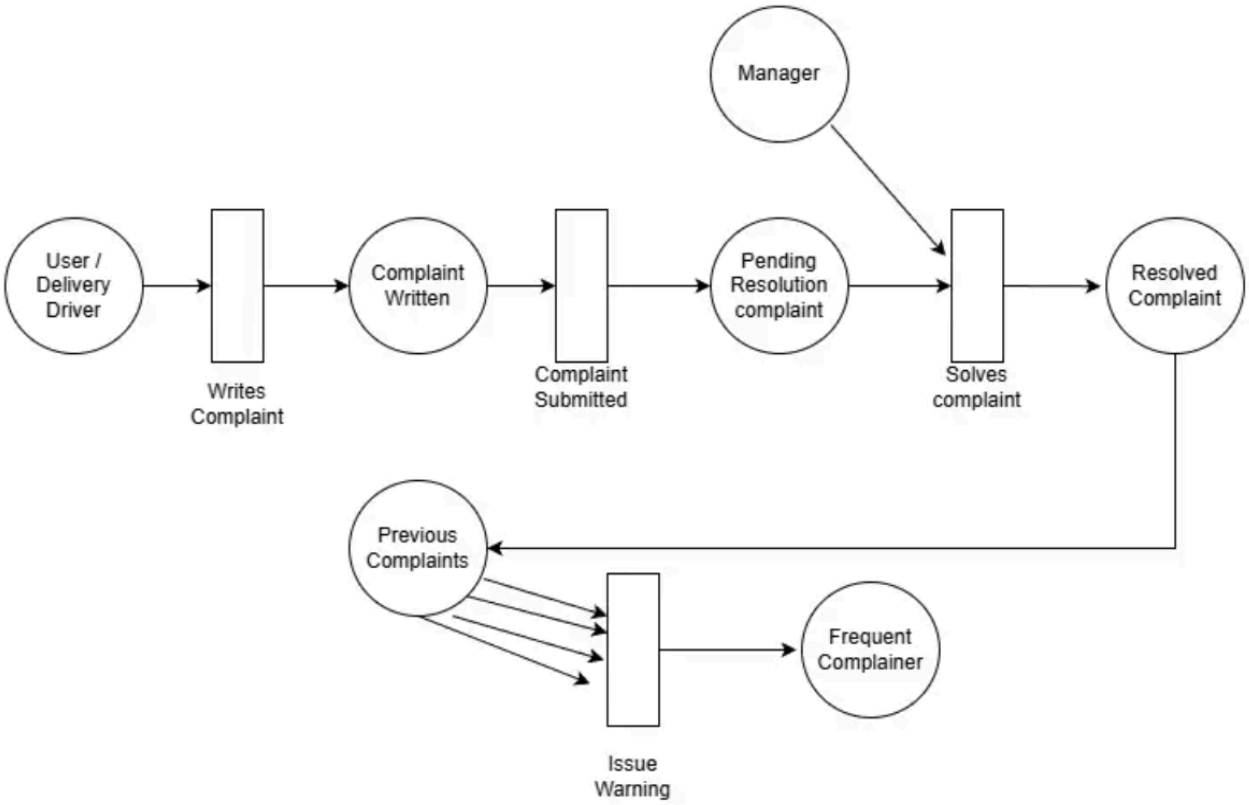
Normal Scenario

1. After an order is marked `DELIVERED`, UI prompts for ratings:
    - Dish rating (per `Dish`)
    - Delivery rating (per `DeliveryPerson`)
2. User submits ratings (1–5 stars + optional comment).
3. `ReputationService` creates `Rating` records and updates aggregated `ReputationRecord` for the `Chef`, `DeliveryPerson`, and optionally the `Customer` (for behavior / fairness).

Exceptional Scenario

- Rating submitted too early (order not `DELIVERED`) → reject.
- Rating already exists for that order → prevent duplicate.

Diagram: Sequence: `UI → ReputationService → RatingRepository, ReputationRepository`.



UC-07: File Complaint (Petri net UC #2)

Actor: Customer / VIP

Normal Scenario

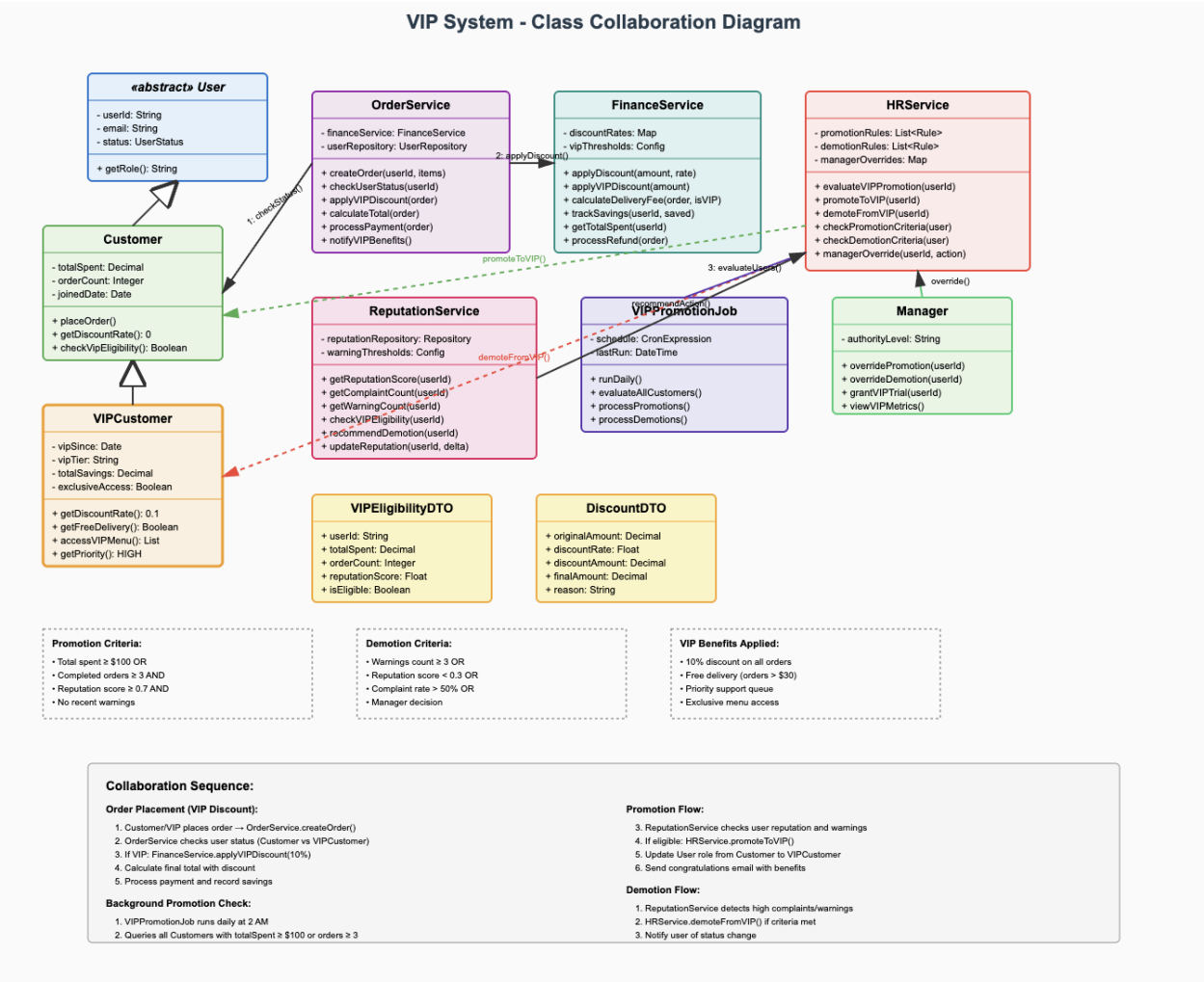
1. User selects an order and clicks “File Complaint”.
2. UI collects complaint type (chef, delivery, system), description, and optional evidence.
3. `ReputationService.fileComplaint()` validates:
    - Order exists and belongs to user.
    - Complaint doesn’t already exist for this order & target.
4. Complaint is stored with status = `PENDING` and assigned to `Manager`.
5. System notifies `Manager` dashboard.

Exceptional Scenarios

- Order not found / not owned by user → error, no complaint created.
- Customer is flagged for abusing complaints (too many frivolous past complaints) → complaint blocked or marked “low priority”, possible warning.

Petri Net Description

- **Places:**
    - `P_ComplaintDraft`
    - `P_ValidComplaint`
    - `P_InvalidComplaint`
    - `P_ComplaintsPending`
    - `P_WarningsIssued`
- **Transitions:**
    - `T_SubmitComplaint` – moves token from `P_ComplaintDraft` to either `P_ValidComplaint` or `P_InvalidComplaint`.
    - `T_QueueComplaint` – from `P_ValidComplaint` to `P_ComplaintsPending`.
    - `T_IssueWarning` – from `P_InvalidComplaint` to `P_WarningsIssued` (if frivolous).
- **Tokens:**
    - Each complaint is a token; flow shows lifecycle.



UC-08: VIP Discount / VIP Promotion

Actor: VIP Customer; Manager (for manual override)

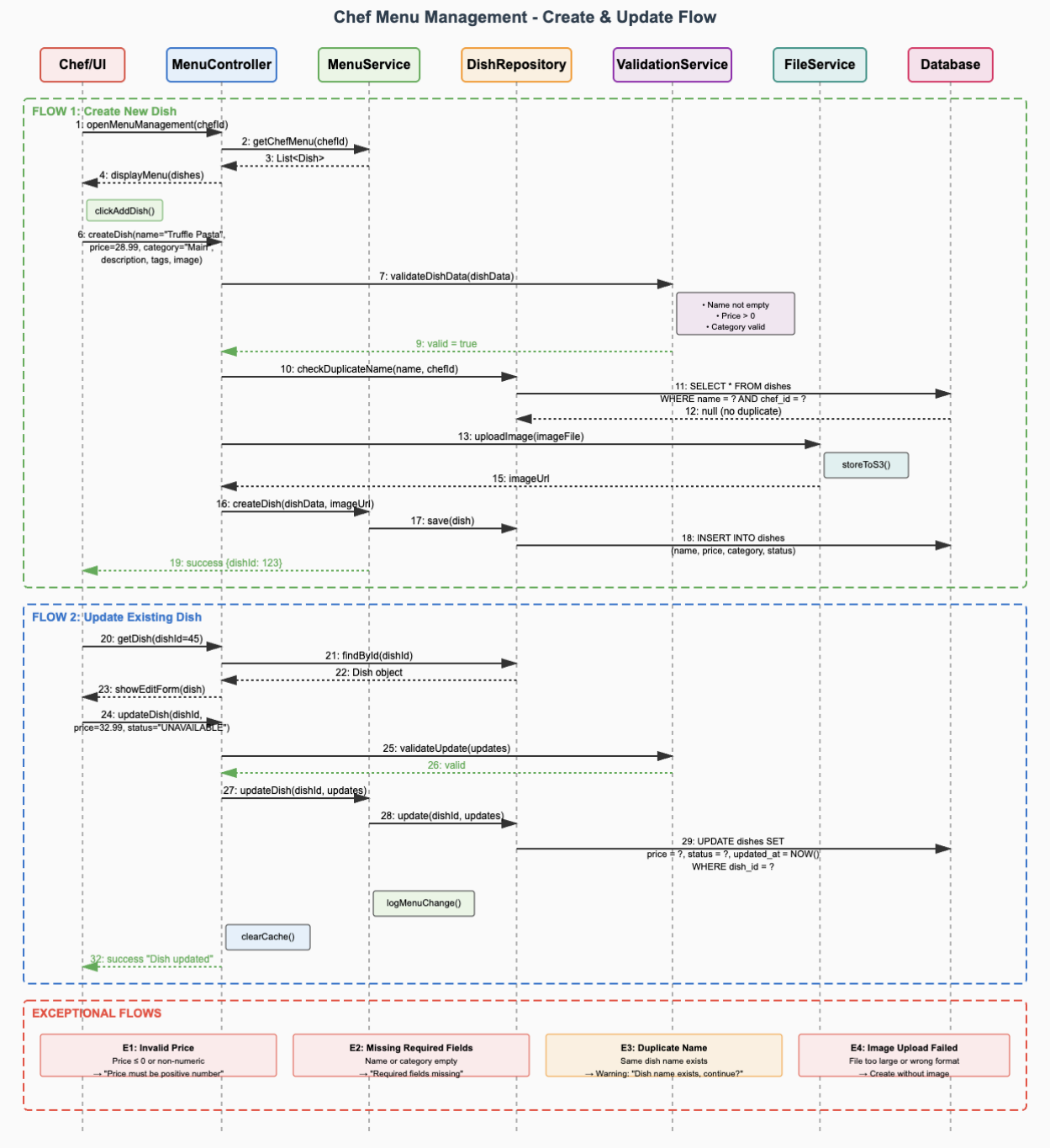
Normal Scenario

1. When `Customer` places an order, `OrderService` checks VIP status.
2. If user is `VIPCustomer`, `FinanceService` applies discount (e.g., % off or free delivery) before payment.
3. Separately, a background rule in `HRService` / `ReputationService` checks total spend and successful orders:
    - If thresholds met (e.g., ≥ $100 or ≥ 3 orders), user is upgraded to `VIPCustomer`.

Exceptional Scenario

- Downgrade: If user receives too many complaints or warnings, `HRService` may demote them back to `Customer`.

Diagram: Class/collaboration focusing on `Customer`, `VIPCustomer`, `OrderService`, `FinanceService`, `HRService`.



UC-09: Create / Update Menu (Chef)

Actor: Chef

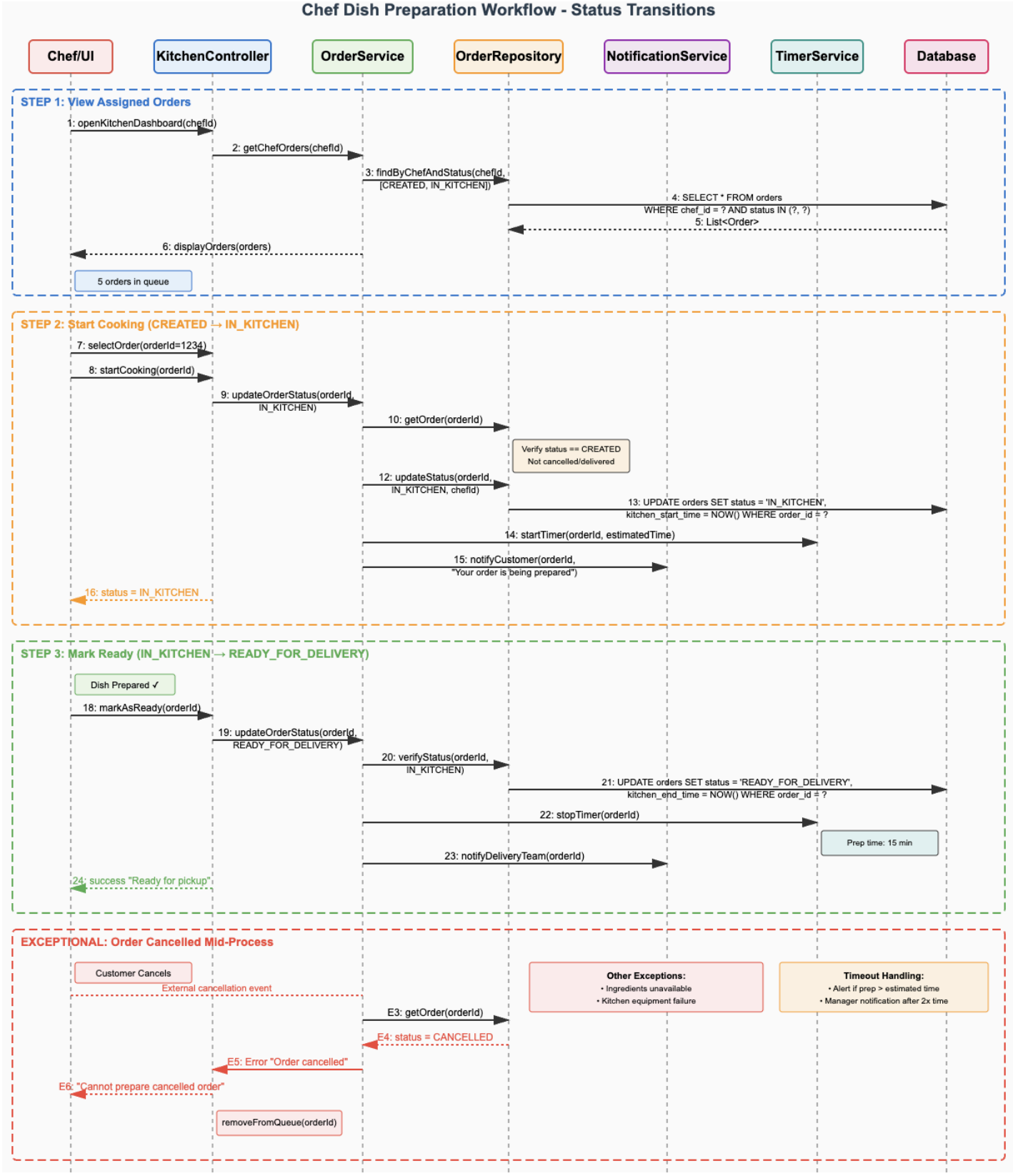
Normal Scenario

1. `Chef` logs in and opens “Manage Menu”.

2. For **Create**:
    - Enters dish name, description, price, category, and upload picture.
    - `MenuService` validates fields, creates `Dish` record (status = ACTIVE).
3. For **Update**:
    - Selects existing dish and edits fields or marks dish UNAVAILABLE.
    - `MenuService` updates the `Dish`.

Exceptional Scenarios

- Invalid price / missing required fields → validation error.
- Dish name duplicate within same restaurant → optional warning.



UC-10: Prepare Dish (Chef)

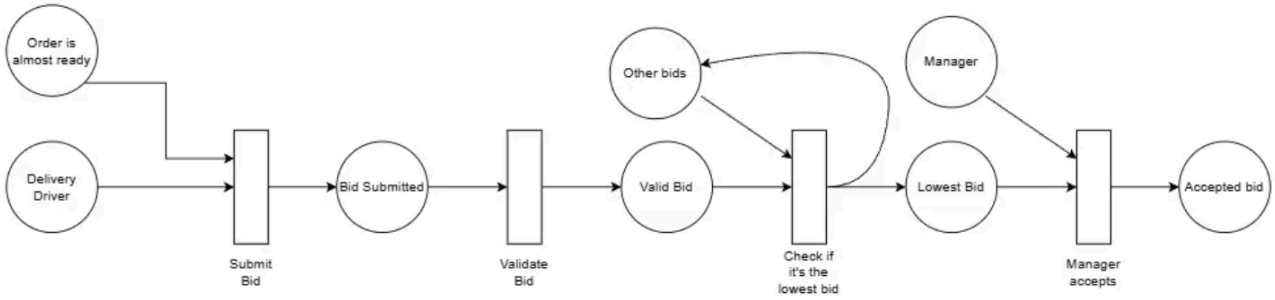
Actor: Chef

Normal Scenario

1. Chef sees list of assigned orders with status `CREATED`.
2. Chef picks one, sets status to `IN_KITCHEN`.
3. After preparation, Chef sets status to `READY_FOR_DELIVERY`.

Exceptional Scenario

- Order cancelled mid-process → status moves to `CANCELLED`; cannot mark as ready.



UC-11: Submit Delivery Bid (Petri net UC #3)

Actor: Delivery Person

Normal Scenario

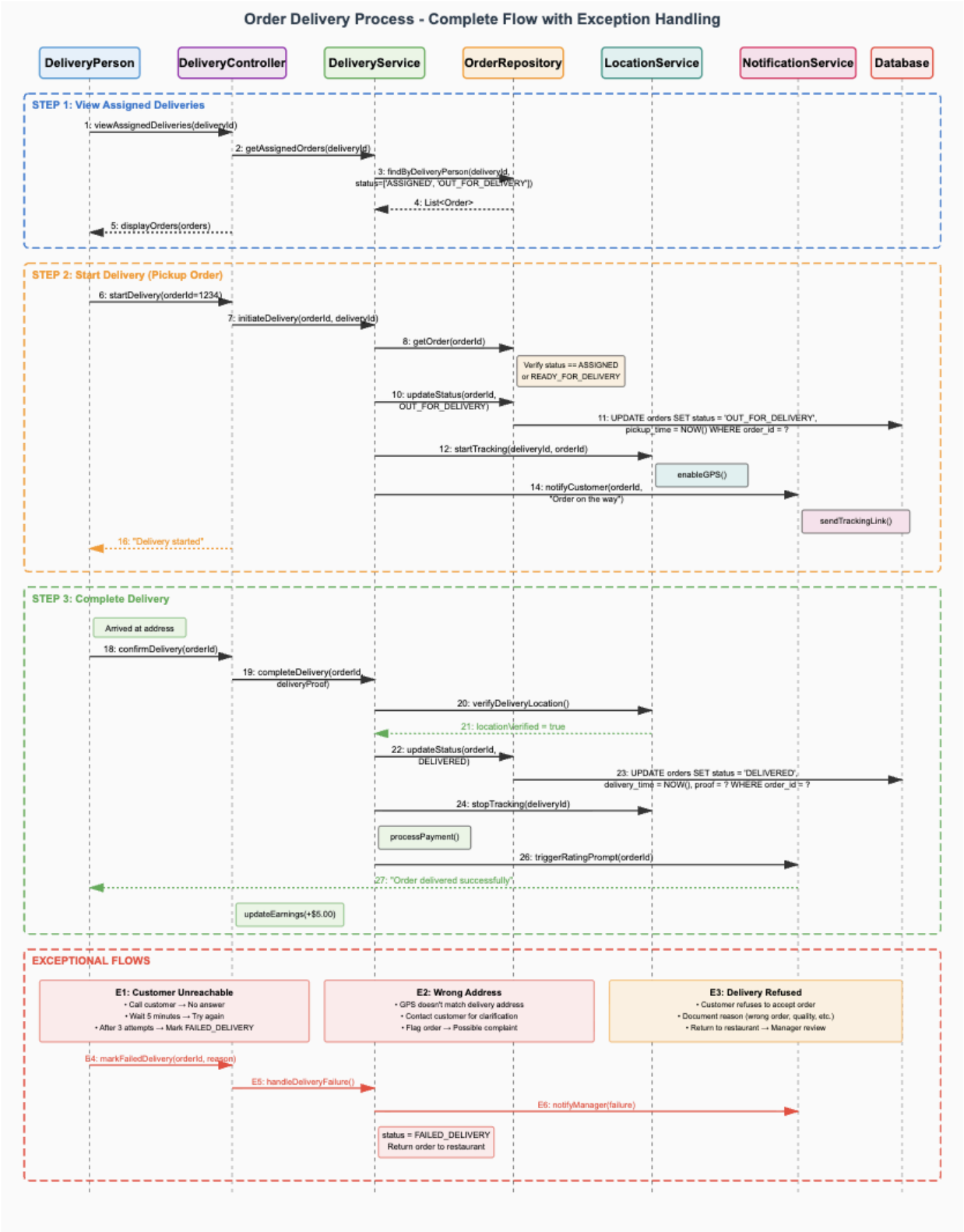
1. DeliveryPerson sees list of orders in `READY_FOR_DELIVERY` with no assigned courier.
2. DeliveryPerson selects an order and enters bid data (ETA, optional fee, willingness).
3. `DeliveryService.submitBid()` validates:
    - Bidding window still open.
    - DeliveryPerson is active and not overloaded.
4. `DeliveryBid` is stored with status = `PENDING`.
5. Manager will later assign based on bids.

Exceptional Scenarios

- Bidding after window closed → bid rejected.
- DeliveryPerson already has too many active deliveries → reject or warn.

Petri Net Description

- **Places:**
    - `P_OrderReadyForBid`
    - `P_EligibleDeliveryPerson`
    - `P_BidSubmitted`
    - `P_BidRejected`
    - `P_BidQueue`
- **Transitions:**
    - `T_SubmitBid` – consumes tokens from `P_OrderReadyForBid` and `P_EligibleDeliveryPerson`, creates `P_BidSubmitted`.
    - `T_ValidateBid` – moves token to either `P_BidQueue` or `P_BidRejected`.



UC-12: Deliver Order

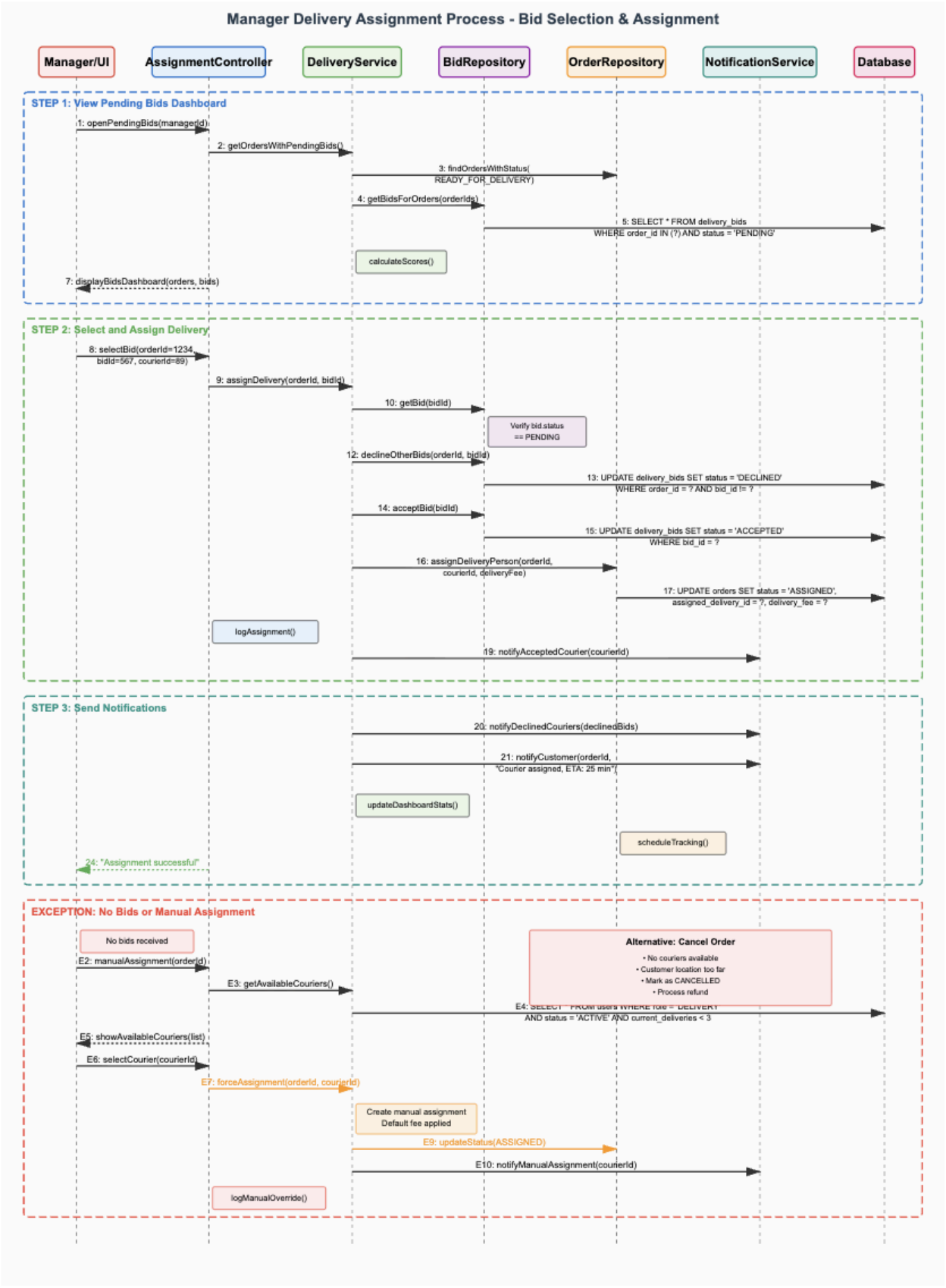
Actor: Delivery Person

Normal Scenario

1. After `Manager` assigns a `DeliveryPerson` to an order, status becomes `ASSIGNED`.
2. DeliveryPerson sees assigned jobs and clicks “Start Delivery”.
3. Status → `OUT_FOR_DELIVERY`.
4. After arriving, DeliveryPerson confirms “Delivered”; status → `DELIVERED`.
5. `DeliveryService` records delivery completion time and triggers rating prompt.

Exceptional Scenarios

- Customer unreachable → mark as `FAILED_DELIVERY` and notify Manager.
- Wrong address → flagged; possible complaint.



UC-13: Assign Deliveries (Manager)

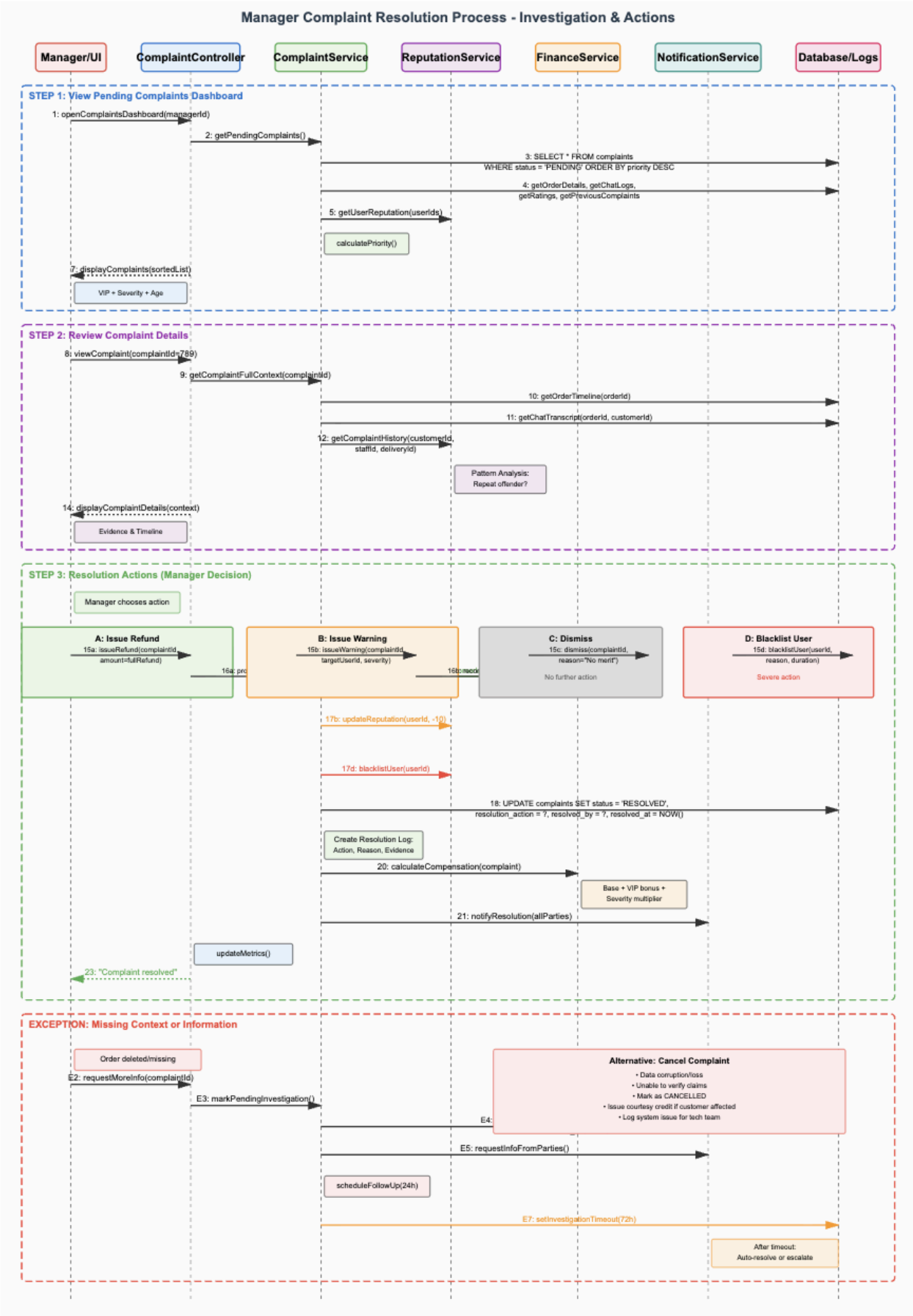
Actor: Manager

Normal Scenario

1. Manager opens “Pending Bids” dashboard.
2. For each order with multiple bids, Manager selects the best based on ETA, rating, tie-break rules.
3. `DeliveryService.assignDelivery(orderId, bidId)`:
    - Locks other bids as `DECLINED`.
    - Links order to chosen `DeliveryPerson`.
    - Sets order status = `ASSIGNED`.

Exceptional Scenario

- No bids received in time window → Manager manually assigns to available DeliveryPerson or cancels order.


Manager Complaint Resolution Process - Investigation & Actions
Manager/UI
ComplaintController
ComplaintService
ReputationService
FinanceService
NotificationService
Database/Logs
STEP 1: View Pending Complaints Dashboard
1: openComplaintsDashboard(managerId)
2: getPendingComplaints()
3: SELECT * FROM complaints
WHERE status = 'PENDING' ORDER BY priority DESC
4: getOrderDetails, getChatLogs,
getRatings, getPreviousComplaints
5: getUserReputation(userIds)
calculatePriority()
7: displayComplaints(sortedList)
VIP + Severity + Age
STEP 2: Review Complaint Details
8: viewComplaint(complaintId=789)
9: getComplaintFullContext(complaintId)
10: getOrderTimeline(orderId)
11: getChatTranscript(orderId, customerId)
12: getComplaintHistory(customerId,
staffId, deliveryId)
Pattern Analysis:
Repeat offender?
14: displayComplaintDetails(context)
Evidence & Timeline
STEP 3: Resolution Actions (Manager Decision)
Manager chooses action
A: Issue Refund
15a: issueRefund(complaintId,
amount=fullRefund)
B: Issue Warning
15b: issueWarning(complaintId,
targetUserId, severity)
C: Dismiss
15c: dismiss(complaintId,
reason="No merit")
No further action
D: Blacklist User
15d: blacklistUser(userId,
reason, duration)
Severe action
17b: updateReputation(userId, -10)
17d: blacklistUser(userId)
18: UPDATE complaints SET status = 'RESOLVED',
resolution_action = ?, resolved_by = ?, resolved_at = NOW()
Create Resolution Log:
Action, Reason, Evidence
20: calculateCompensation(complaint)
Base + VIP bonus +
Severity multiplier
21: notifyResolution(allParties)
updateMetrics()
23: "Complaint resolved"
EXCEPTION: Missing Context or Information
Order deleted/missing
E2: requestMoreInfo(complaintId)
E3: markPendingInvestigation()
Alternative: Cancel Complaint
• Data corruption/loss
• Unable to verify claims
• Mark as CANCELLED
• Issue courtesy credit if customer affected
• Log system issue for tech team
E5: requestInfoFromParties()
scheduleFollowUp(24h)
E7: setInvestigationTimeout(72h)
After timeout:
Auto-resolve or escalate

UC-14: Resolve Complaints (Manager)

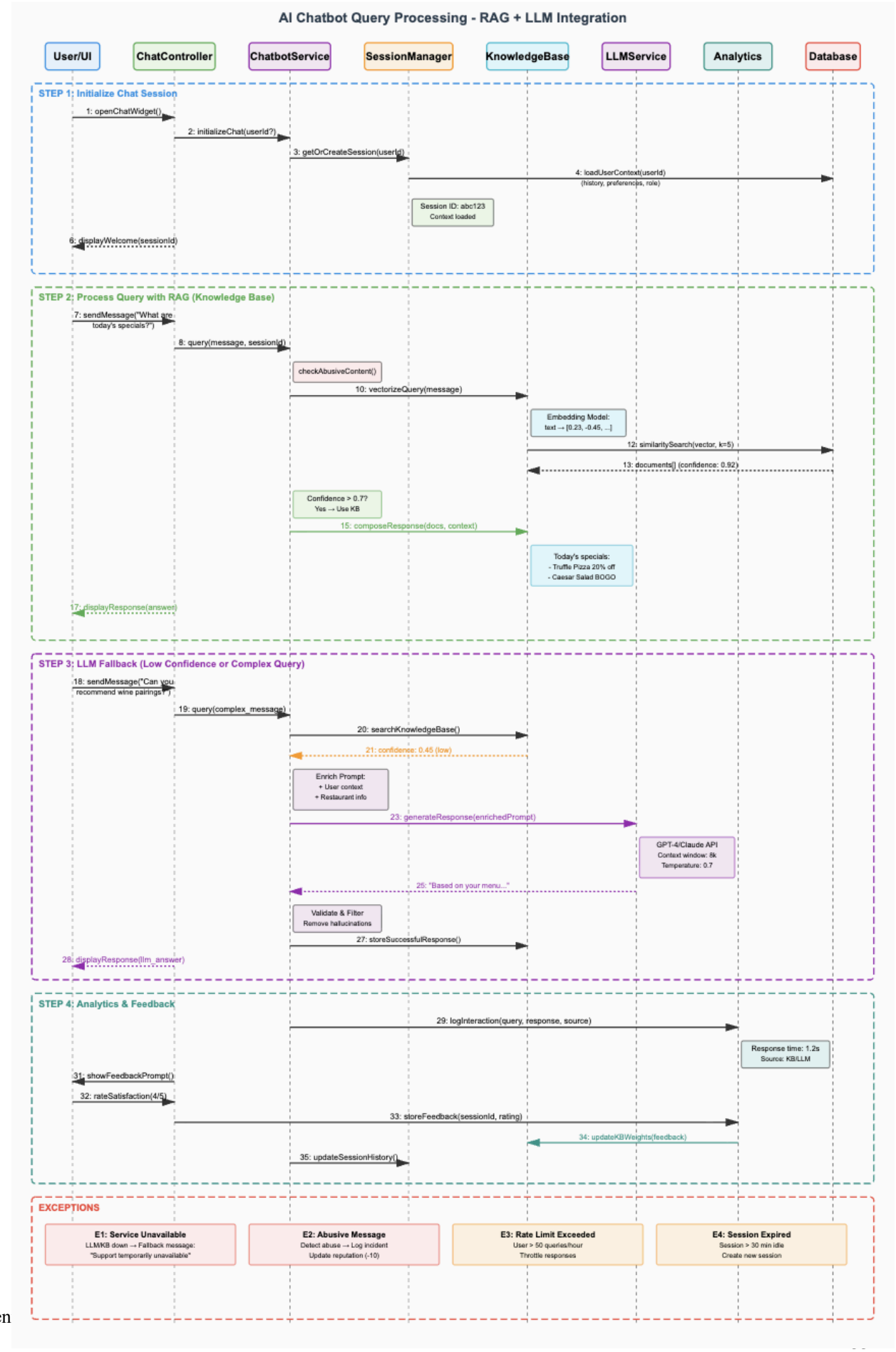
Actor: Manager

Normal Scenario

1. Manager opens “Complaints” dashboard with `PENDING` complaints.
2. For each complaint, Manager reviews:
    - Order details, chat logs, ratings, previous complaints.
3. Manager chooses action:
    - Dismiss complaint (mark `RESOLVED_NO_ACTION`).
    - Issue warning to staff or customer (record `ReputationRecord` change).
    - Refund part of order via `FinanceService`.
    - Blacklist a user if multiple serious complaints.
4. System updates complaint status and reputation/wallet as appropriate.

Exceptional Scenario

- Complaint has missing context (order deleted incorrectly) → Manager requests extra info; complaint remains `PENDING` or moves to `CANCELLED`.


AI Chatbot Query Processing - RAG + LLM Integration
User/UI
ChatController
ChatbotService
SessionManager
KnowledgeBase
LLMService
Analytics
Database
STEP 1: Initialize Chat Session
1: openChatWidget()
2: initializeChat(userId?)
3: getOrCreateSession(userId)
4: loadUserContext(userId)
(history, preferences, role)
Session ID: abc123
Context loaded
6: displayWelcome(sessionId)
STEP 2: Process Query with RAG (Knowledge Base)
7: sendMessage("What are today's specials?")
8: query(message, sessionId)
checkAbusiveContent()
10: vectorizeQuery(message)
Embedding Model:
text → [0.23, -0.45, ...]
12: similaritySearch(vector, k=5)
13: documents[] (confidence: 0.92)
Confidence > 0.7?
Yes → Use KB
15: composeResponse(docs, context)
Today's specials:
- Truffle Pizza 20% off
- Caesar Salad BOGO
17: displayResponse(answer)
STEP 3: LLM Fallback (Low Confidence or Complex Query)
18: sendMessage("Can you recommend wine pairings")
19: query(complex_message)
20: searchKnowledgeBase()
21: confidence: 0.45 (low)
Enrich Prompt:
+ User context
+ Restaurant info
23: generateResponse(enrichedPrompt)
GPT-4/Claude API
Context window: 8k
Temperature: 0.7
25: "Based on your menu..."
Validate & Filter
Remove hallucinations
27: storeSuccessfulResponse()
28: displayResponse(llm_answer)
STEP 4: Analytics & Feedback
29: logInteraction(query, response, source)
Response time: 1.2s
Source: KB/LLM
31: showFeedbackPrompt()
32: rateSatisfaction(4/5)
33: storeFeedback(sessionId, rating)
34: updateKBWeights(feedback)
35: updateSessionHistory()
EXCEPTIONS
E1: Service Unavailable
LLM/KB down → Fallback message:
"Support temporarily unavailable"
E2: Abusive Message
Detect abuse → Log incident
Update reputation (-10)
E3: Rate Limit Exceeded
User > 50 queries/hour
Throttle responses
E4: Session Expired
Session > 30 min idle
Create new session

Confiden

UC-15: Query AI Chatbot

Actor: Any user (Visitor, Customer, VIP, staff)

Normal Scenario

1. User opens chat widget and enters a question (e.g., “What are today’s specials?”, “Where’s my order?”).
2. `ChatbotService.query()`:
    - Creates `ChatbotSession` if needed.
    - Searches `KnowledgeBaseEntry` (RAG) for relevant info.
    - If answer found, compose reply.
    - Otherwise, forwards enriched prompt to LLM.
3. System shows natural language response; optionally logs feedback.

Exceptional Scenarios

- LLM or KB unavailable → show fallback message “Support is temporarily unavailable, please contact staff.”
- User sends abusive messages → trigger reputation/blacklist logic.

3.2 Supplementary Requirements

Security:

- All passwords hashed and stored securely.
- JWT tokens used for authentication.
- Only managers can access HR and complaint modules.

Performance:

- The system must handle up to 100 users with average response under 2 seconds.

Usability:

- Interface supports mobile and desktop layouts.
- AI chatbot available on all pages.

Reliability:

- All financial transactions use rollback protection.

- Orders automatically saved on system failure.

Maintainability:

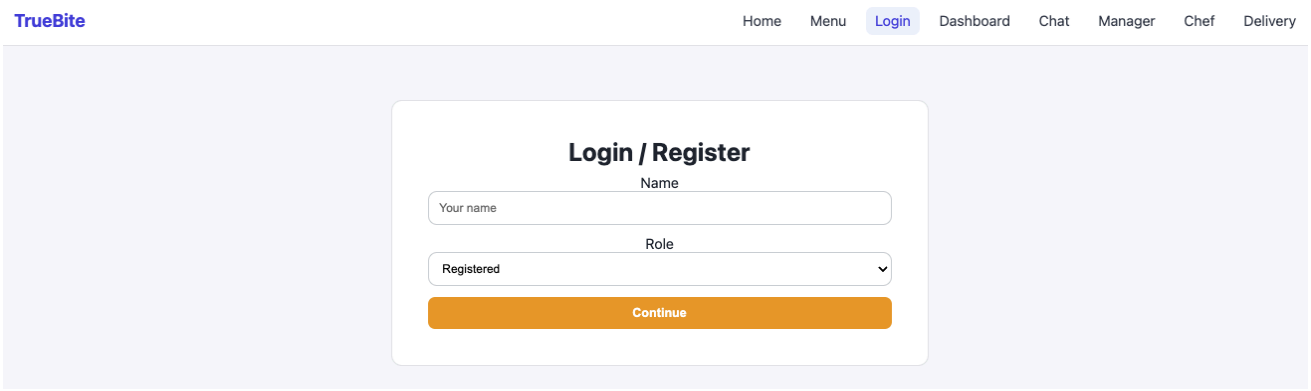
- Backend organized into modular Flask APIs.
- Clear code documentation for future updates.

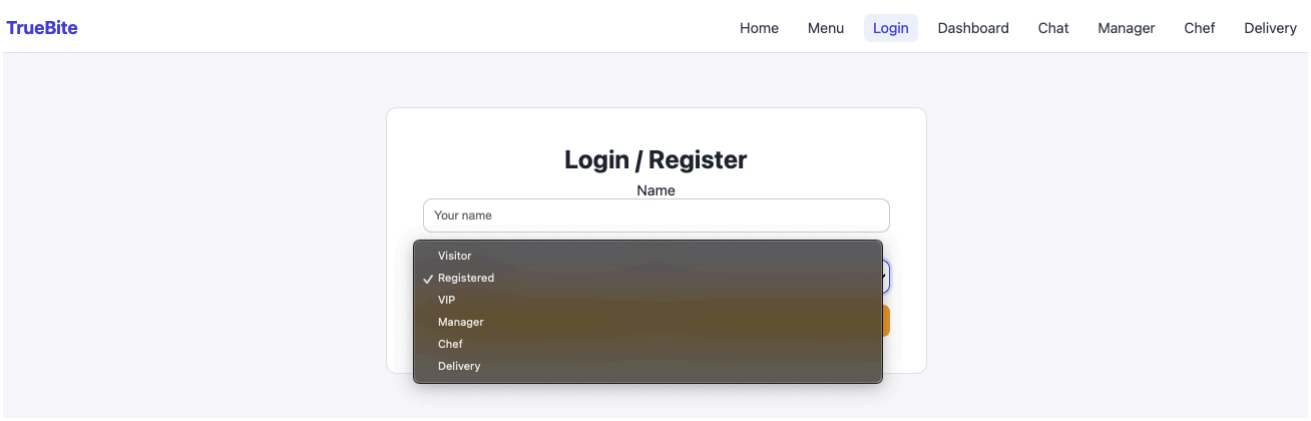
4. Supporting Information

4.1 Appendix A: GUI Mockups

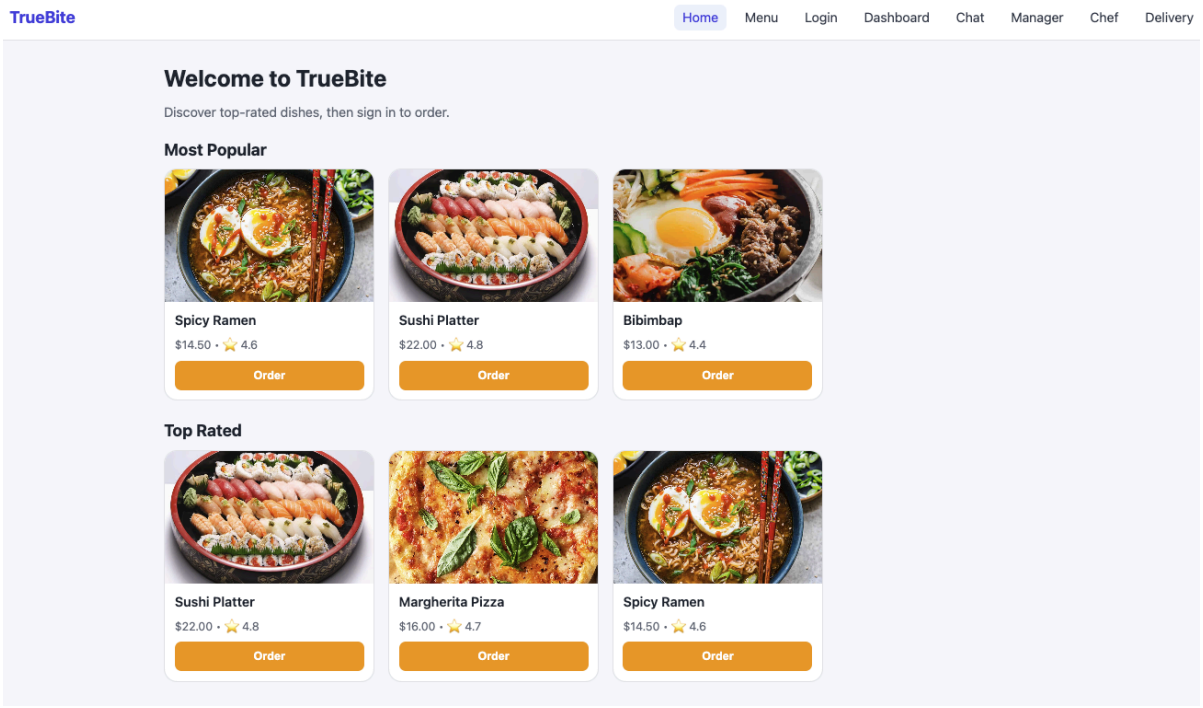
This section includes wireframes or screenshots illustrating:

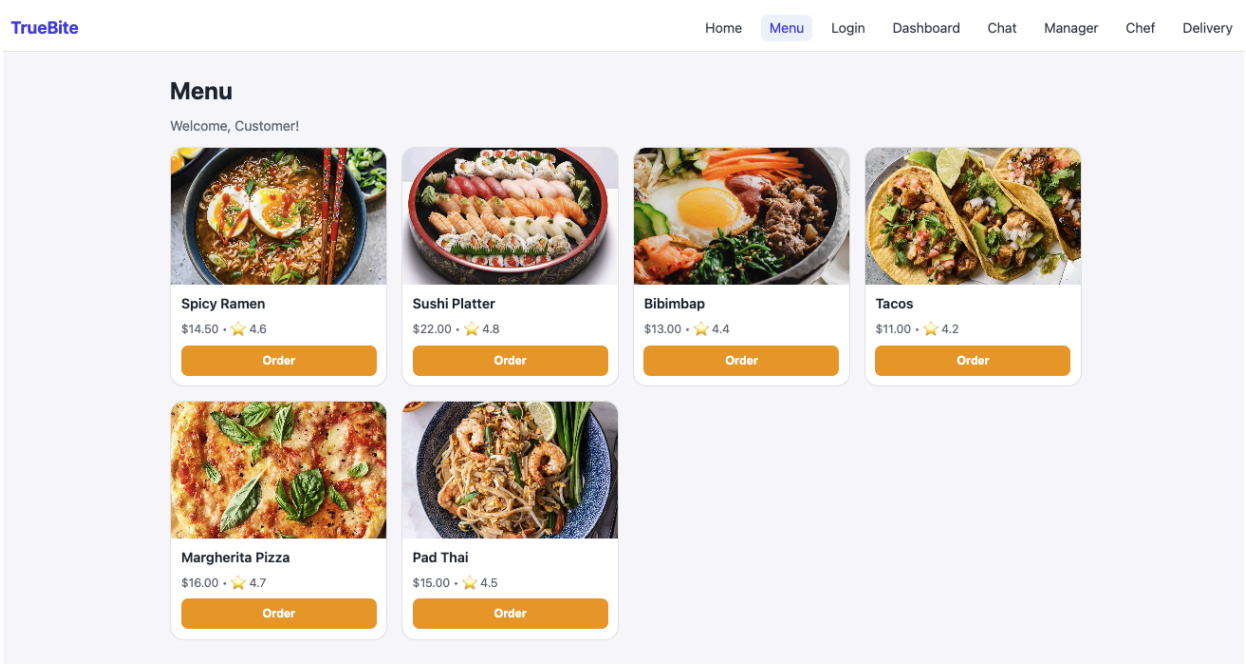
- Login and Registration Pages → These screens allow users to authenticate or choose a role for demonstration purposes.Users can proceed as a Customer, Chef, Delivery Driver, or Manager, aligning with the system's four primary roles.



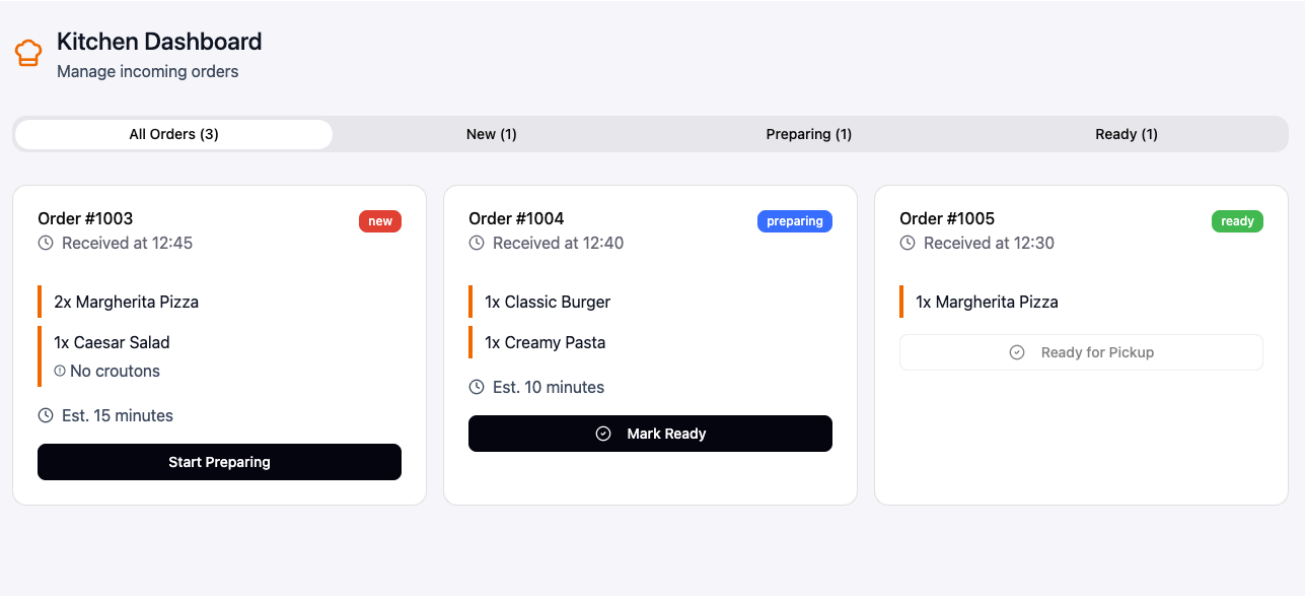


- Customer Dashboard and Menu Browsing Interface → displays available dishes with photos, prices, star ratings, and an Order button placeholder. Additional features such as shopping cart and filtering will be implemented in later phases, but the current screen demonstrates the primary customer browsing experience.

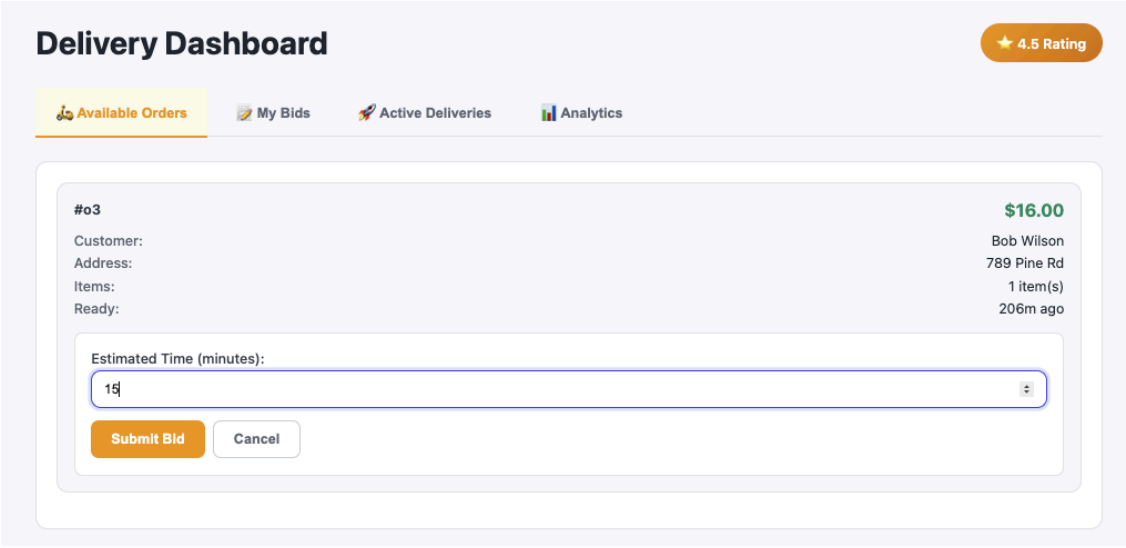


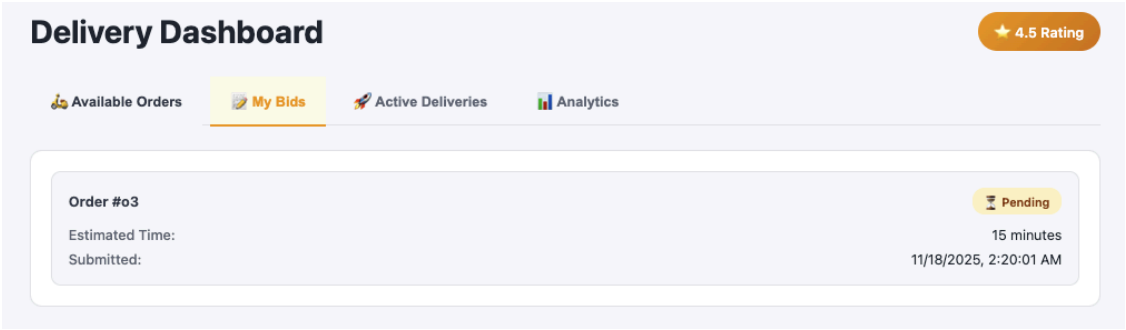


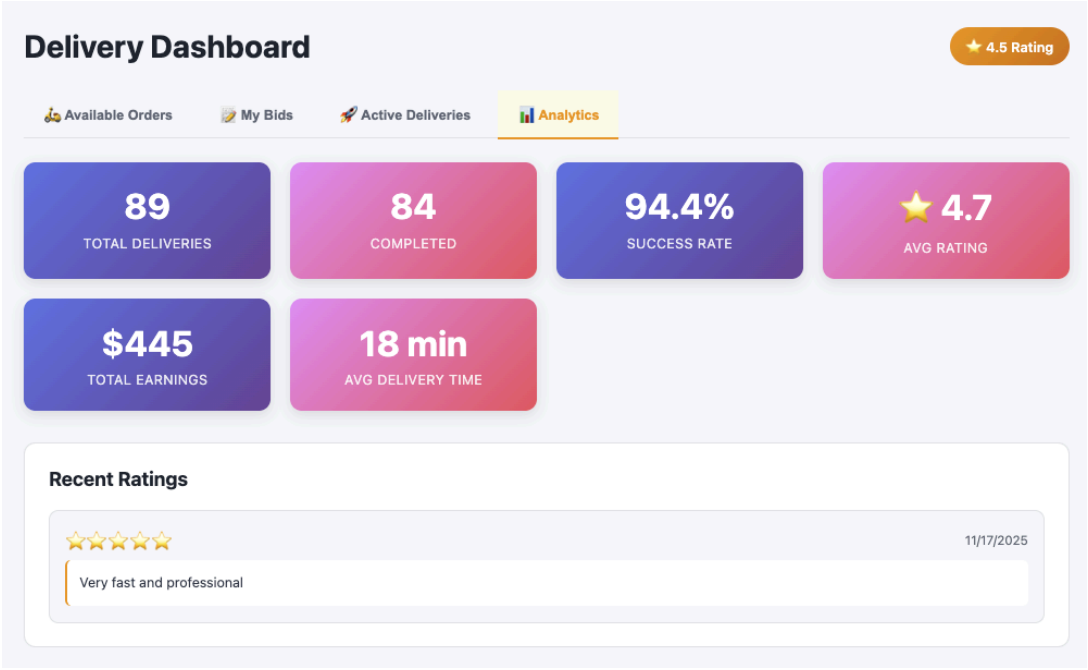
- Chef Dashboard - Order Management → allows kitchen staff to view and manage incoming orders. Orders are grouped into tabs such as **New**, **Preparing**, and **Ready**, each displayed as clean, minimal cards showing order number, time received, ordered items, special notes, estimated prep time, and action buttons such as **Start Preparing** and **Mark Ready.**



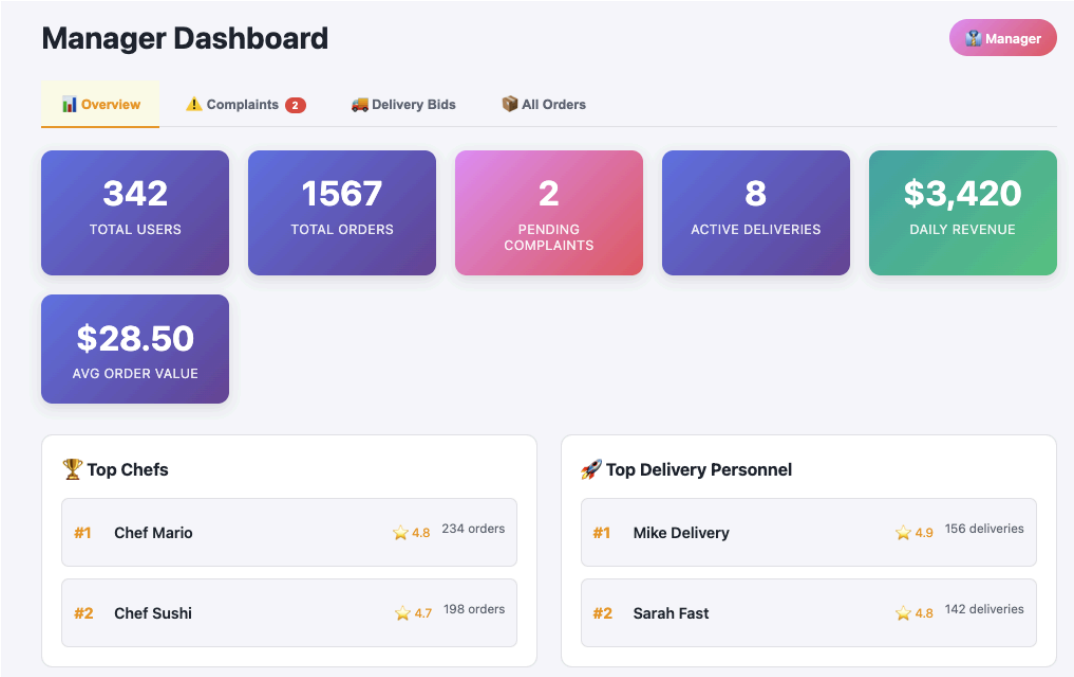
- Delivery Driver Dashboard → delivery personnel can view available delivery requests, view active assigned deliveries, enter a fee and ETA to submit delivery bids, view active assigned deliveries, mark deliveries as completed, and track personal reputation and warnings. This dashboard is divided into **Available Deliveries**, **Active Deliveries**, and **Earnings/Stats** sections.

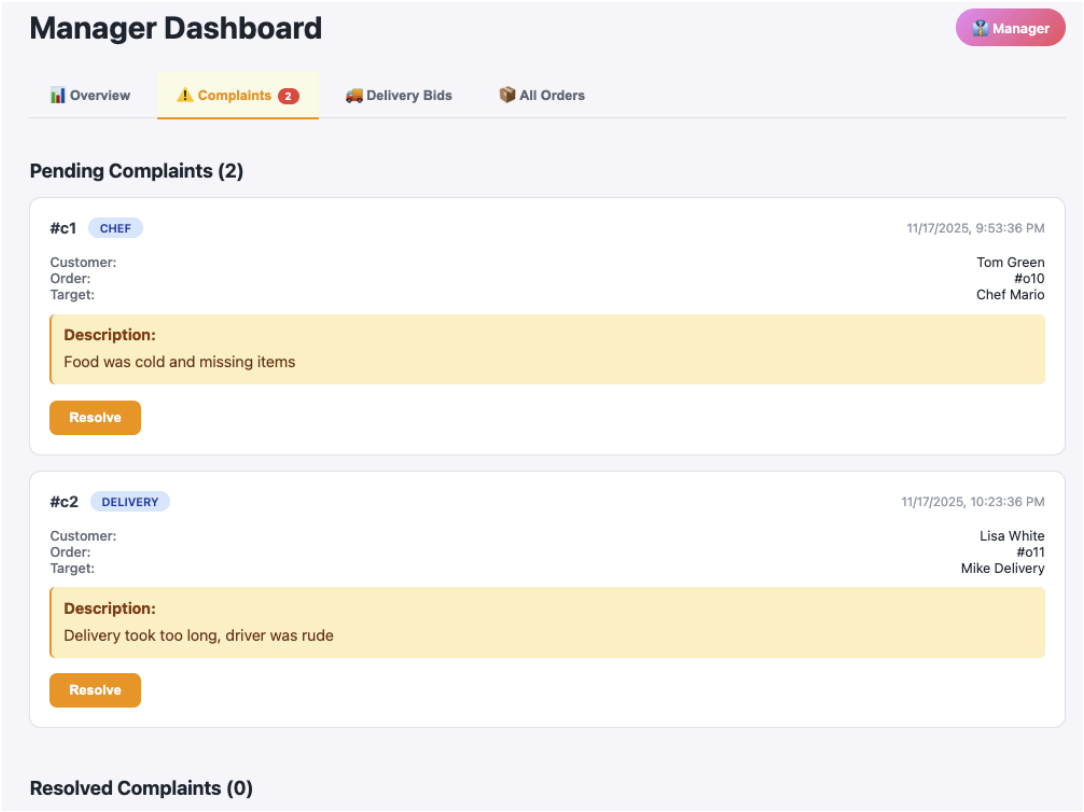


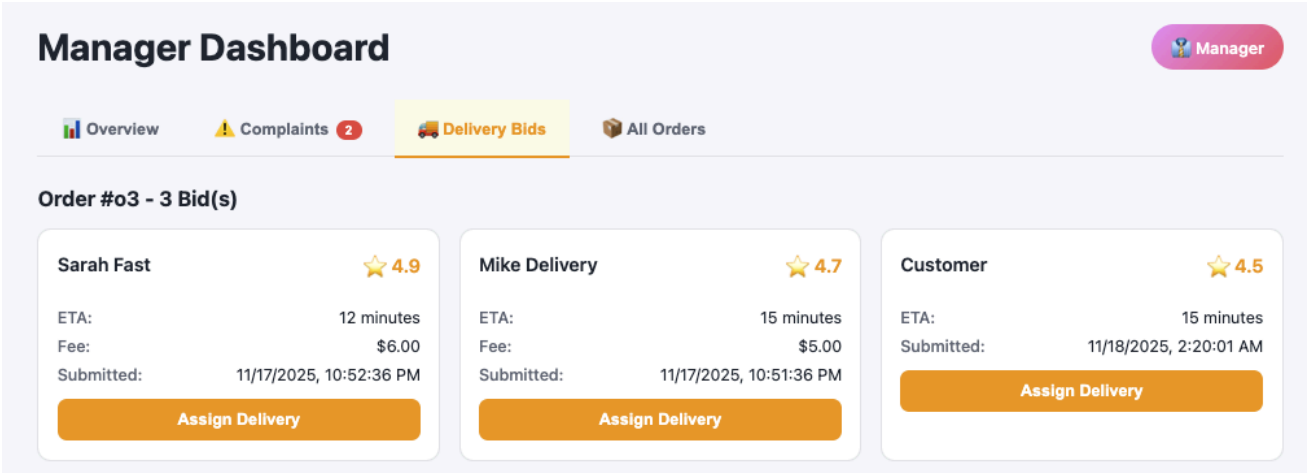


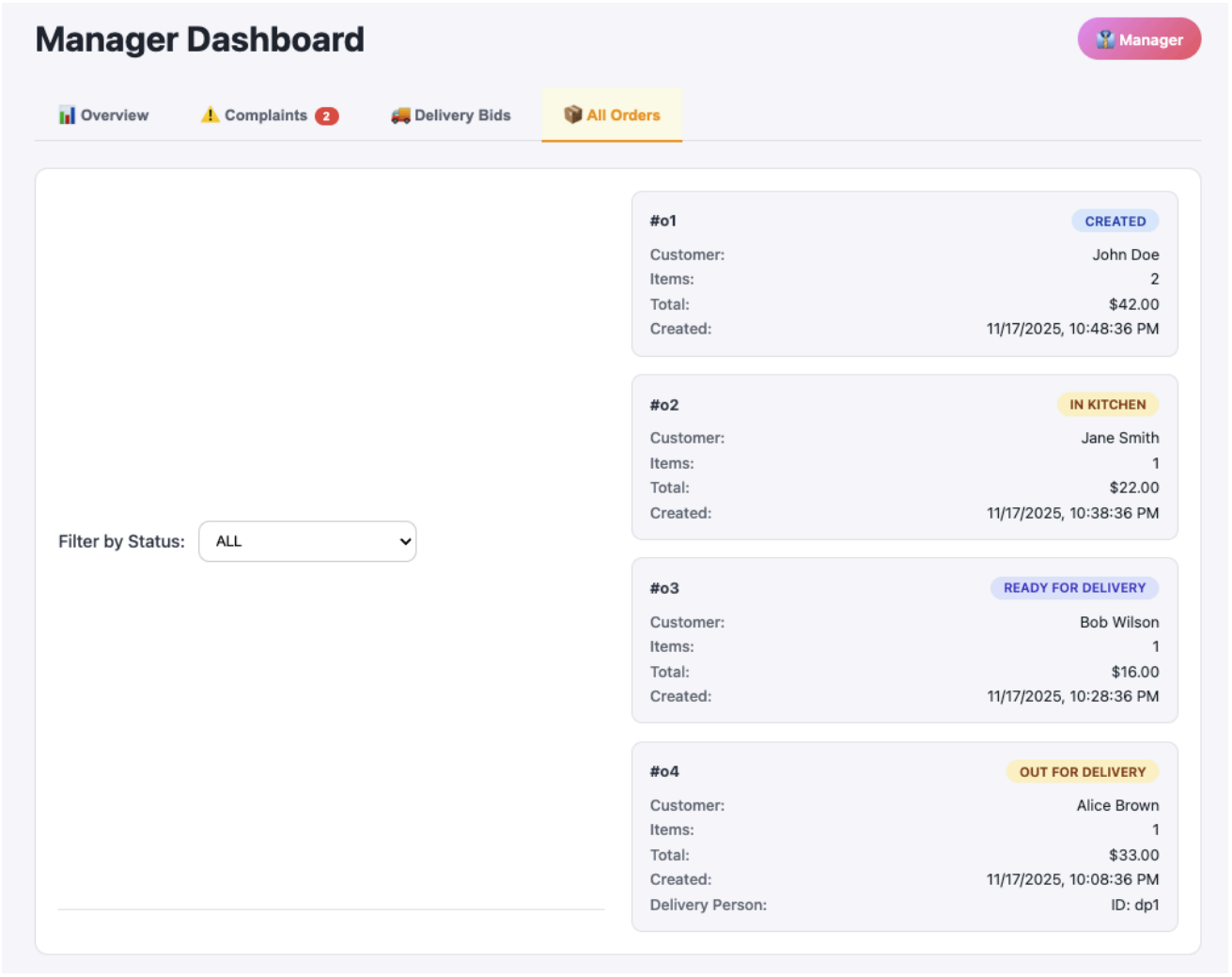


- Manager Dashboard → centralizes key administrative actions such as reviewing delivery bids, resolving customer complaints, and monitoring staff performance and warnings. It provides managers with a clear overview of system activity and tools for maintaining service quality.

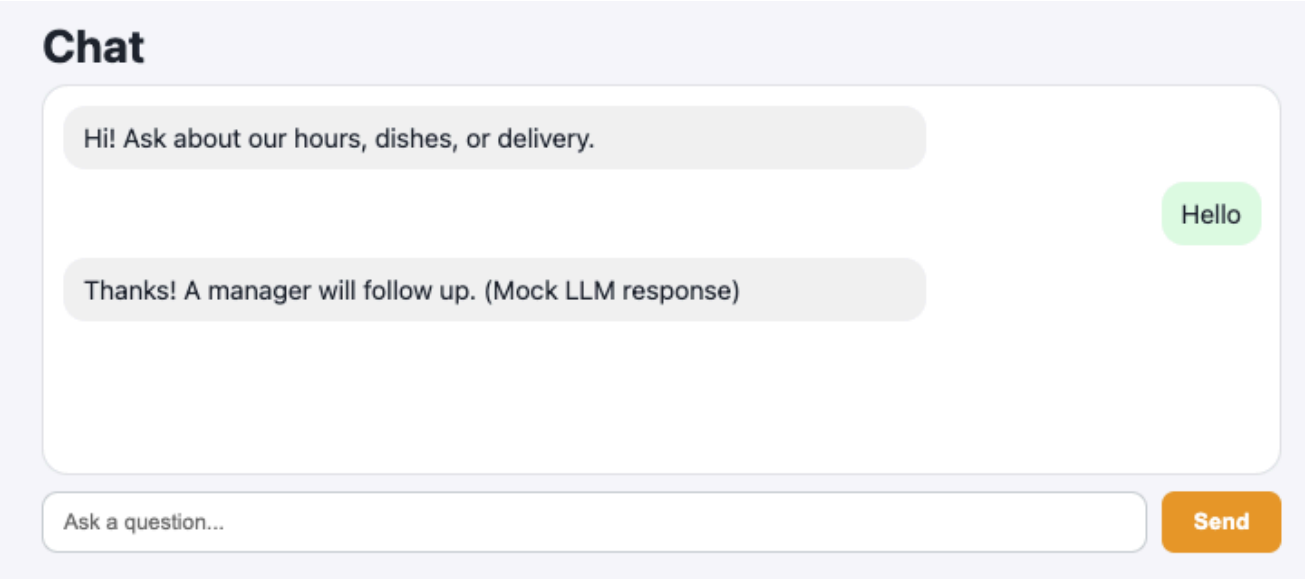





Manager Dashboard
Manager
Overview
Complaints 2
Delivery Bids
All Orders
Order #o3 - 3 Bid(s)
Sarah Fast
4.9
ETA: 12 minutes
Fee: $6.00
Submitted: 11/17/2025, 10:52:36 PM
Assign Delivery
Mike Delivery
4.7
ETA: 15 minutes
Fee: $5.00
Submitted: 11/17/2025, 10:51:36 PM
Assign Delivery
Customer
4.5
ETA: 15 minutes
Submitted: 11/18/2025, 2:20:01 AM
Assign Delivery


Manager Dashboard
Manager
Overview
Complaints 2
Delivery Bids
All Orders
Filter by Status: ALL
#o1
CREATED
Customer: John Doe
Items: 2
Total: $42.00
Created: 11/17/2025, 10:48:36 PM
#o2
IN KITCHEN
Customer: Jane Smith
Items: 1
Total: $22.00
Created: 11/17/2025, 10:38:36 PM
#o3
READY FOR DELIVERY
Customer: Bob Wilson
Items: 1
Total: $16.00
Created: 11/17/2025, 10:28:36 PM
#o4
OUT FOR DELIVERY
Customer: Alice Brown
Items: 1
Total: $33.00
Created: 11/17/2025, 10:08:36 PM
Delivery Person: ID: dp1

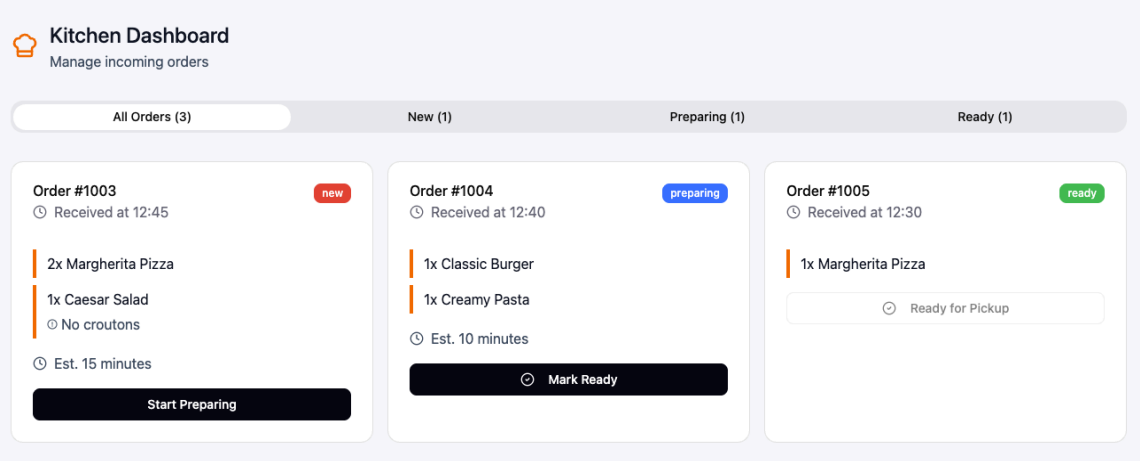
- Chatbot Interface → allows users to interact with the system through a simple text input field, with responses displayed in a scrollable message history. Messages are styled to clearly distinguish between system and user responses, creating an intuitive conversation flow.

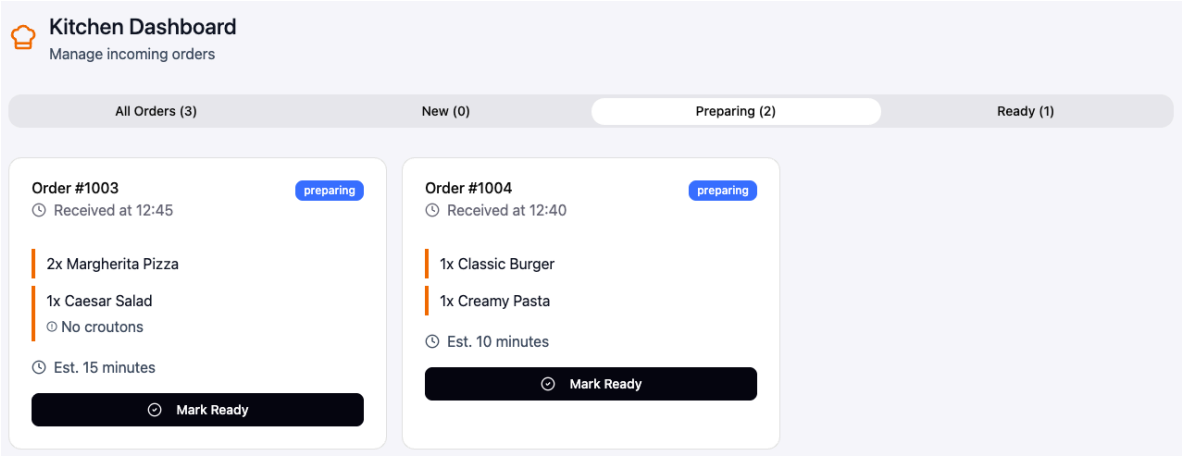


- Prototype Flow — Example: Chef Marks Order as Ready

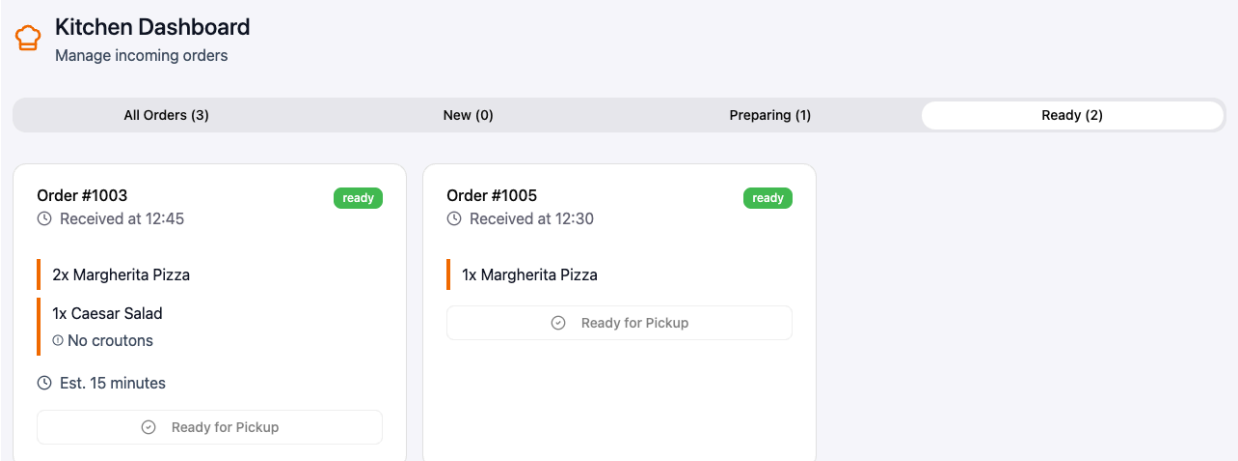
The example shown is a Chef updating an order status:

1. Chef selects a new order from the **New** tab to start preparing. Order moves into the **Preparing** tab.

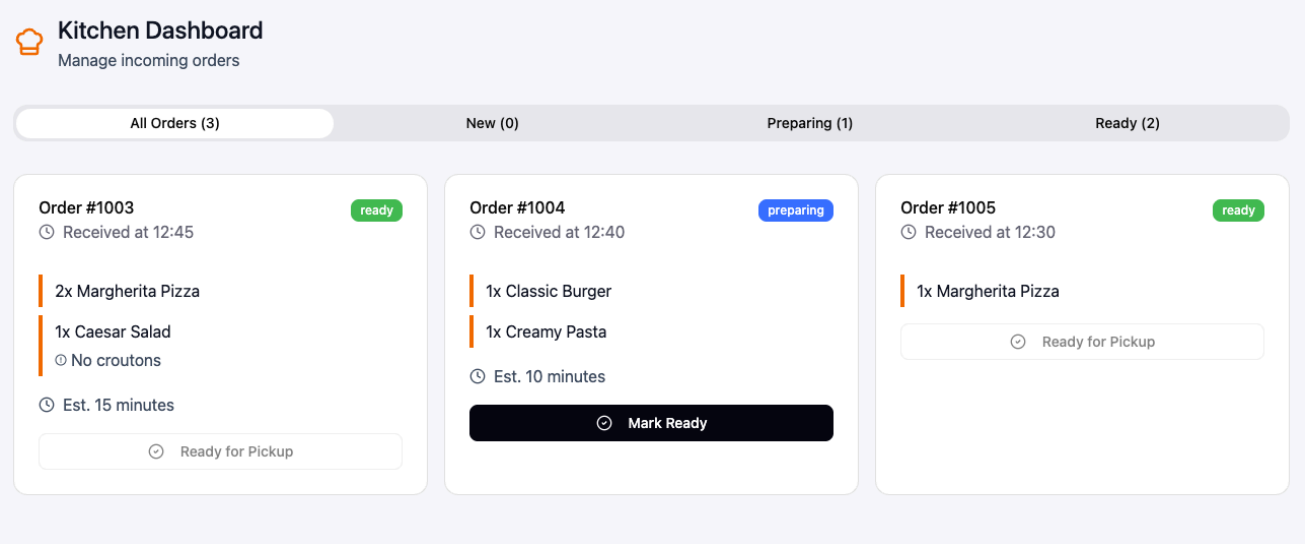




2. Once the order is ready, the chef can mark it as ready. The order moves from the **Preparing** tab into the **Ready** tab.



3. Order is now marked as **Ready**, and the card moves into the '**Ready**' section under **All Orders**.



4.2 Appendix B: Memo & Notes

Memo – Group Meetings and Team Coordination
Date: November 18, 2025
To: Professor Jie Wei
From: Team TrueBite (Arsenii, Angus, Diana, Nick, Bek)

Over the course of Phase II, our group met several times and coordinated mainly via Discord and a shared GitHub repository.

- We held two live online meetings:
    - November 8, 9:00–10:00 PM – finalized class diagram, ER schema, and core use cases.
    - November 12, 9:00–10:30 PM – divided work for Petri nets, sequence diagrams, and GUI prototypes.
- We also had one short in-person meeting after class on November 6 to align on the architecture and assign service-layer responsibilities.

Due to different class schedules, jobs, and commuting times, synchronous meetings were limited. Most of the collaboration happened asynchronously:

- We used a dedicated Discord channel for daily check-ins and code/diagram reviews.
- Each member owned a set of deliverables (e.g., ER + DB schema, diagrams, pseudo-code, or GUI mockups) and pushed updates to GitHub for review.

Challenges:

- Coordination around midterms caused delays; several members had overlapping exams and project deadlines in other courses.

- Aligning diagram conventions (naming, multiplicities, states) across people took extra time to avoid inconsistencies.

Despite these challenges, all members contributed meaningfully. We adjusted task assignments when someone was overloaded and used GitHub issues to track what remained. The report and design reflect input from all team members.

4.3 Links/Materials

GitHub: https://github.com/bshiribaiev/TrueBite