**YogeshMela Project Report**

An online Qurbani Market

**Objectives**

* Deliver a functional livestock marketplace that connects verified sellers with buyers through secure, auditable workflows.
* Provide end-to-end listing and order management with optional delivery and butcher services subject to admin oversight.
* Offer an admin capability to verify sellers, moderate content, manage promotions, and oversee operations.
* Establish a maintainable Laravel MVC architecture with normalized data models, seeders, and documentation to support future growth.

**Introduction**

**YogeshMela** is a Laravel-based web application designed to streamline animal trade with trust and usability. The system targets three primary roles: Buyer, Seller, and Admin. Buyers discover listings, filter by animal attributes, and place orders; Sellers create and manage listings and promotions.

Admins verify sellers, review promotions, and monitor activity. The repository structure reflects clear separation of concerns: models and controllers in app/, Blade views in resources/views, web flows in routes, and database schema in database/migrations plus factories/seeders for reproducible data.

Documentation in Documentations/ captures Activity, Class, ERD, DFD, Sequence, Use Case, and Gantt artifacts, with Goru.md consolidating domain assumptions and feature notes. The UI spans home, listings (index, show), authentication (login, register), dashboards for buyer, seller, and admin, seller listing management (create/edit), promotion attachment, checkout, and cart. The schema covers users, listings, orders, deliveries, butcher orders, promotions, sessions, and carts, mapping directly to the domain processes.

**Methodology**

**Requirement Analysis**

Requirements were derived from marketplace norms. Core needs include robust user roles and authorization, listing metadata (animal type, breed, age, weight, price, location, vaccination), browsing with filters, order placement, optional logistics and processing services, seller verification, and promotional visibility. Non-functional requirements emphasize security (auth, role gating), data integrity, performance at browse scale, and documentation for onboarding.

**Feasibility**

Laravel was selected for accelerated development through expressive routing, Eloquent ORM relationships, migration-based schema control, and Blade templating for server-rendered pages. This stack reduces the operational complexity of a SPA while preserving SEO and accessibility. Seeders and factories enable demo data and deterministic testing.

**Analysis**

The domain model centers on Users, Listings, and Orders with optional attached services: Deliveries and Butcher Orders (0..1 per order). Promotions attach to Listings and undergo admin review. Roles (buyer, seller, admin, butcher, delivery man) allow differentiated dashboards and permissions. Relationships are explicit in migrations and reflected in model associations; cardinalities align with marketplace logic: a seller has many listings; a buyer has many orders; a listing can receive many orders; an order may include at most one delivery and at most one butcher order.

**Design**

The application adopts MVC architecture:

* Models encapsulate data and relationships (users, listings, orders, deliveries, butcher orders, promotions).
* Controllers coordinate validation, persistence, and view selection, keeping business logic focused.
* Views in Blade render pages with filter controls, forms, dashboards, and details.
* Routes partition public pages, authenticated flows, and admin tools.
* Documentation mirrors the architecture: Activity diagrams outline user and admin flows; Use Case diagrams formalize actor-system interactions with includes (mandatory login for ordering and dashboards) and extends (optional filters and service selections); Sequence diagrams trace MVC and entity lifelines across browsing, ordering, and approvals; the Class diagram establishes domain structure; the ERD shows keys and cardinalities; the DFD illustrates external actors, processes, and stores; the Gantt plan aligns tasks across SDLC phases.

**Implementation**

Migrations define normalized tables with foreign keys and status enums. Factories generate realistic test data; seeders populate the environment. Controllers implement flows for authentication, listing CRUD, order creation, service attachments, promotions, and admin approvals. Views provide buyer listing discovery, seller listing creation/editing, admin moderation, cart and checkout screens, and role dashboards. The cart and session migrations support buyer workflows and sign-in persistence.

**Integration**

Cross-feature integrations include:

* Orders binding to Listings and Buyers, with optional service records tied by foreign keys.
* Promotions visible in seller and admin views.
* Dashboards rendering role-specific aggregates and actions.
* Checkout bridging cart data to order creation and payment choice.
* Integration testing focuses on coherent navigation, data consistency, and role-based access control.

**Testing**

Planned tests validate user registration/login, listing index filtering, order creation with optional services, admin verification and promotion decisions, and dashboard visibility. Factories and seeders standardize test fixtures. Edge cases include unauthorized access, invalid status transitions, and foreign key constraints.

**Bugfix**

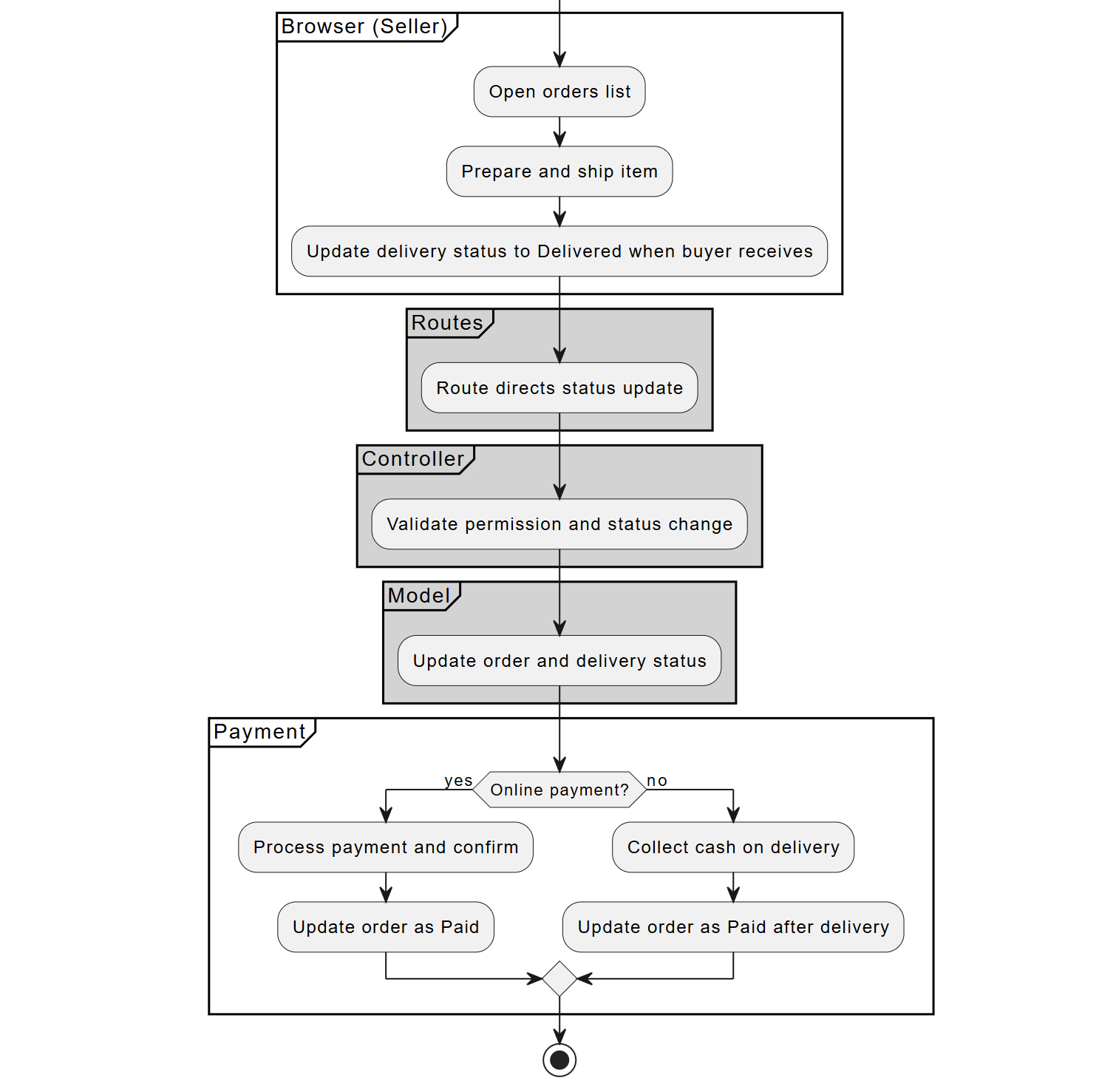
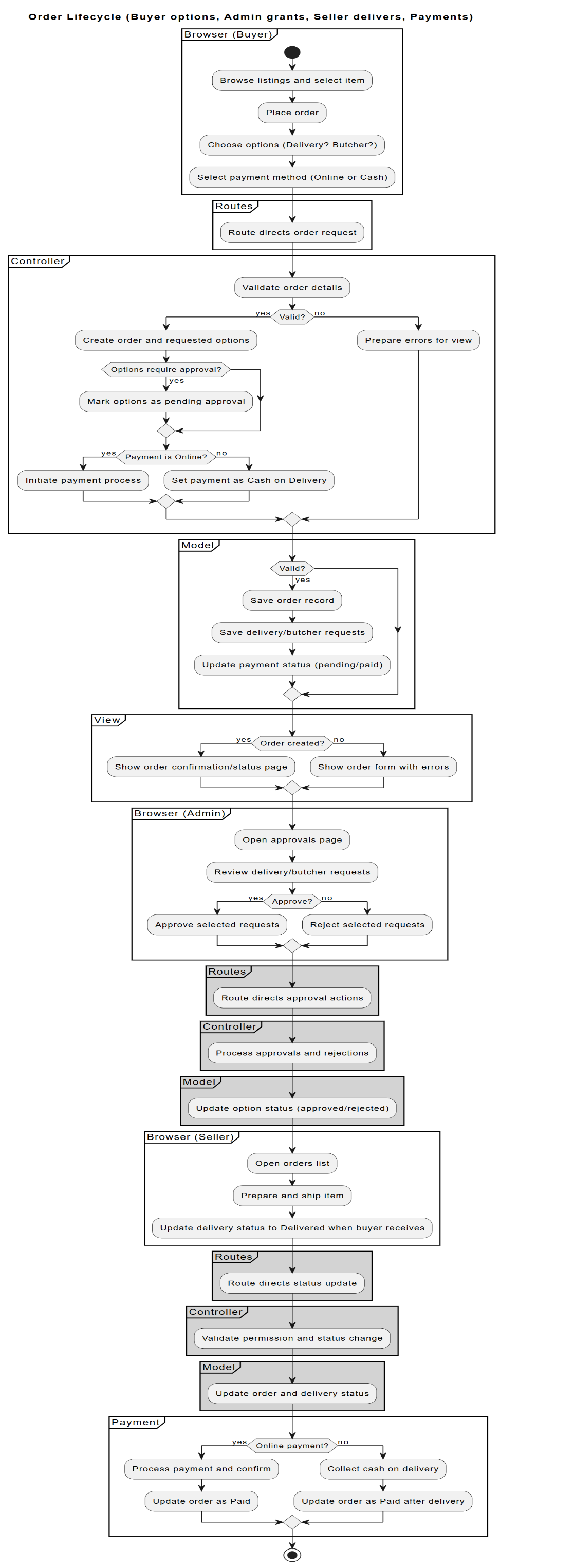
Defects are triaged by path criticality and user impact: authentication and listing browse are prioritized, followed by order and service attachments. Iterative fixes strengthen validation, error messaging, and null-safety around optional services.

**Release**

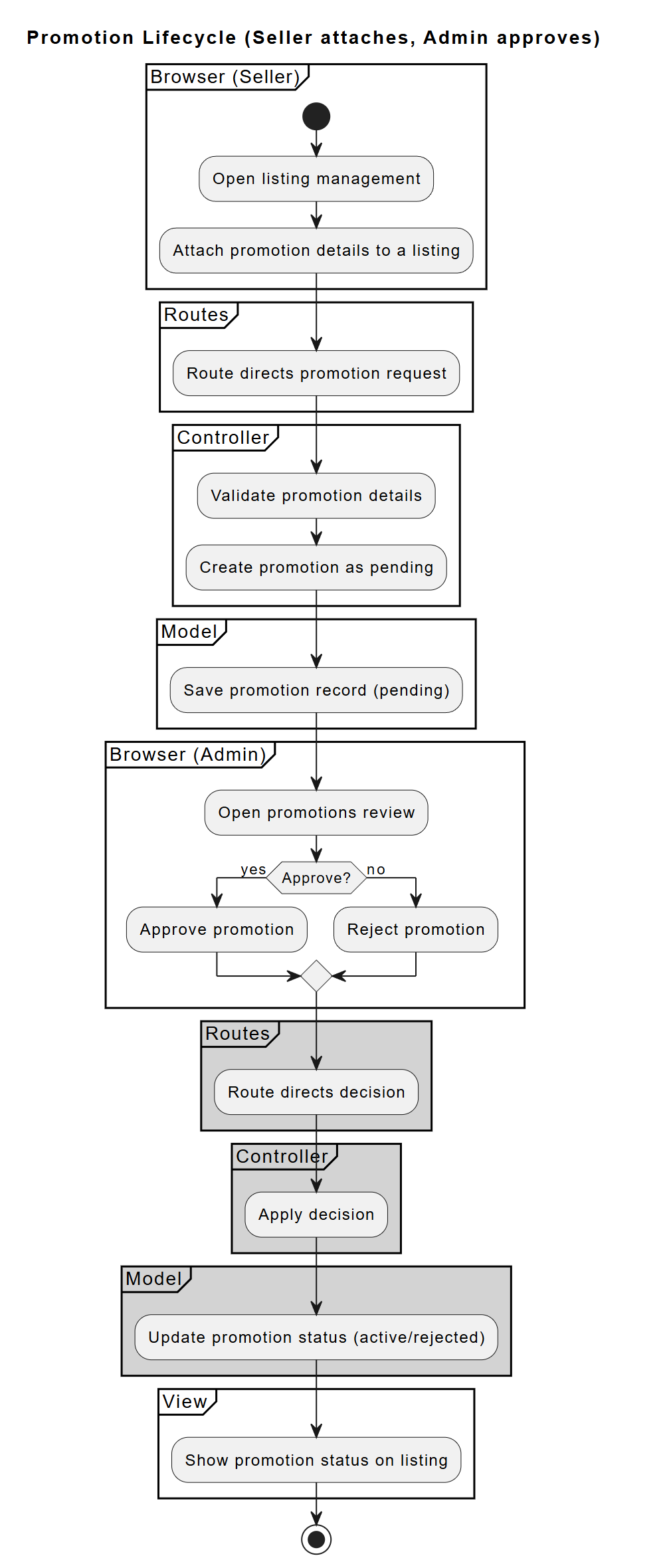
The short timeline drives incremental releases tied to Gantt phases. Milestones include design complete, feature complete, and release readiness. Documentation and diagrams are updated continuously to match implementation.

**Diagrams**

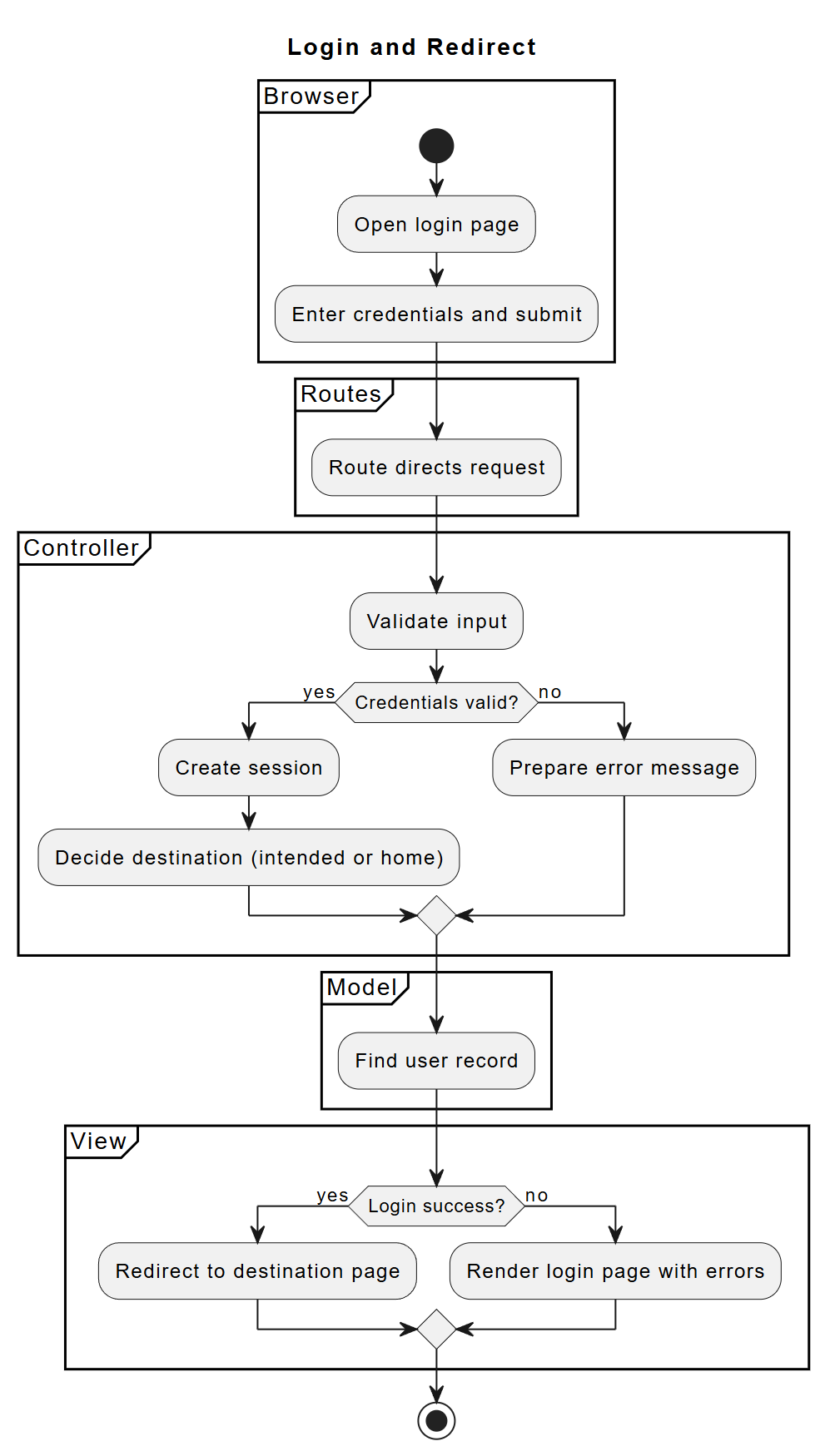
**Activity diagram** summarizes top flows: login/redirect; registration; the order lifecycle from browse to delivered, including admin approvals for optional delivery/butcher and seller marking delivered; seller verification; promotions attach-and-approve; dashboards; and seller add listing.



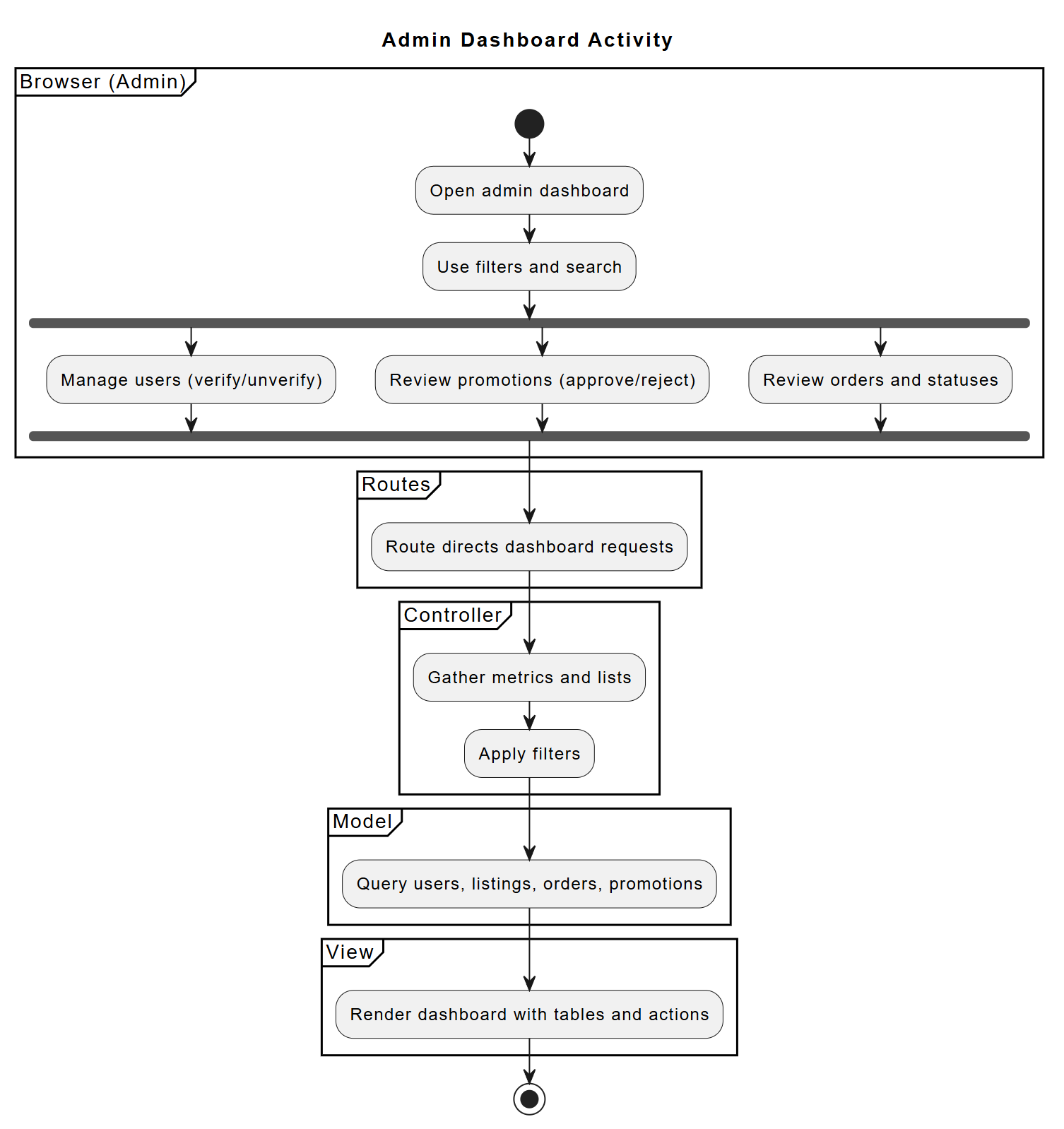
**Figure 1: Activity Diagram for Order LifeCycle**



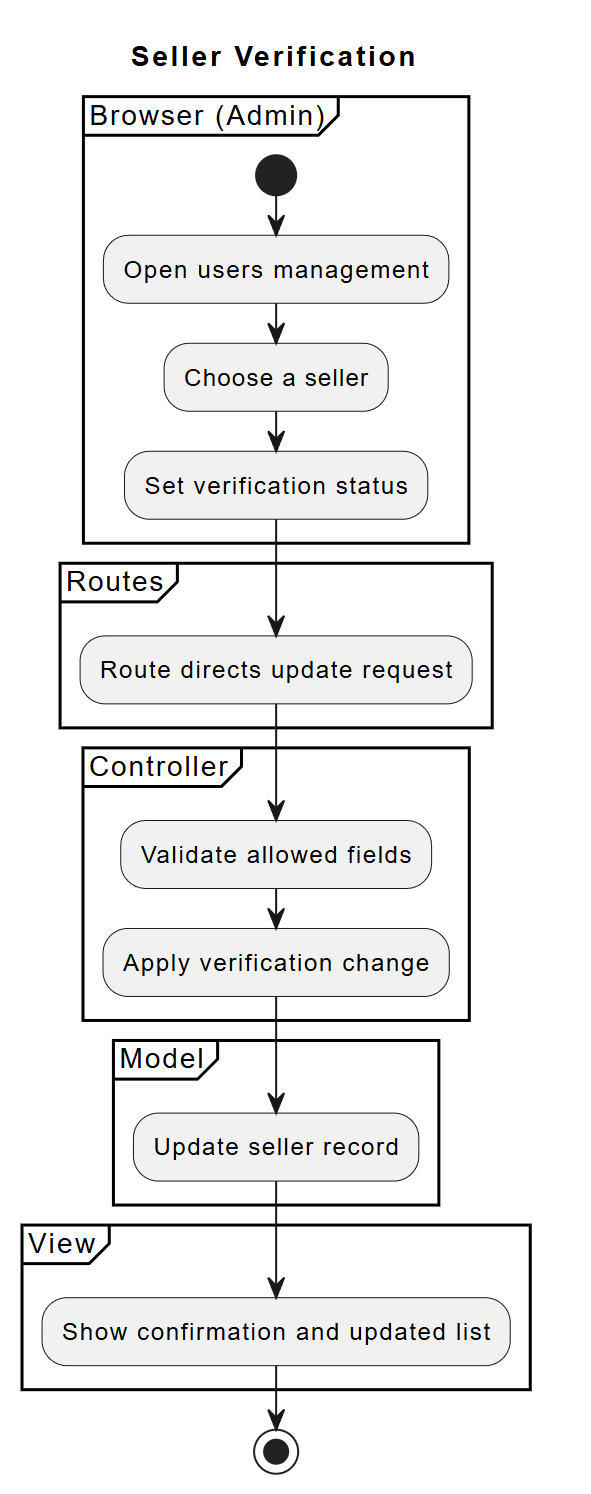
**Figure 2: Activity Diagram for Promotion of listings**



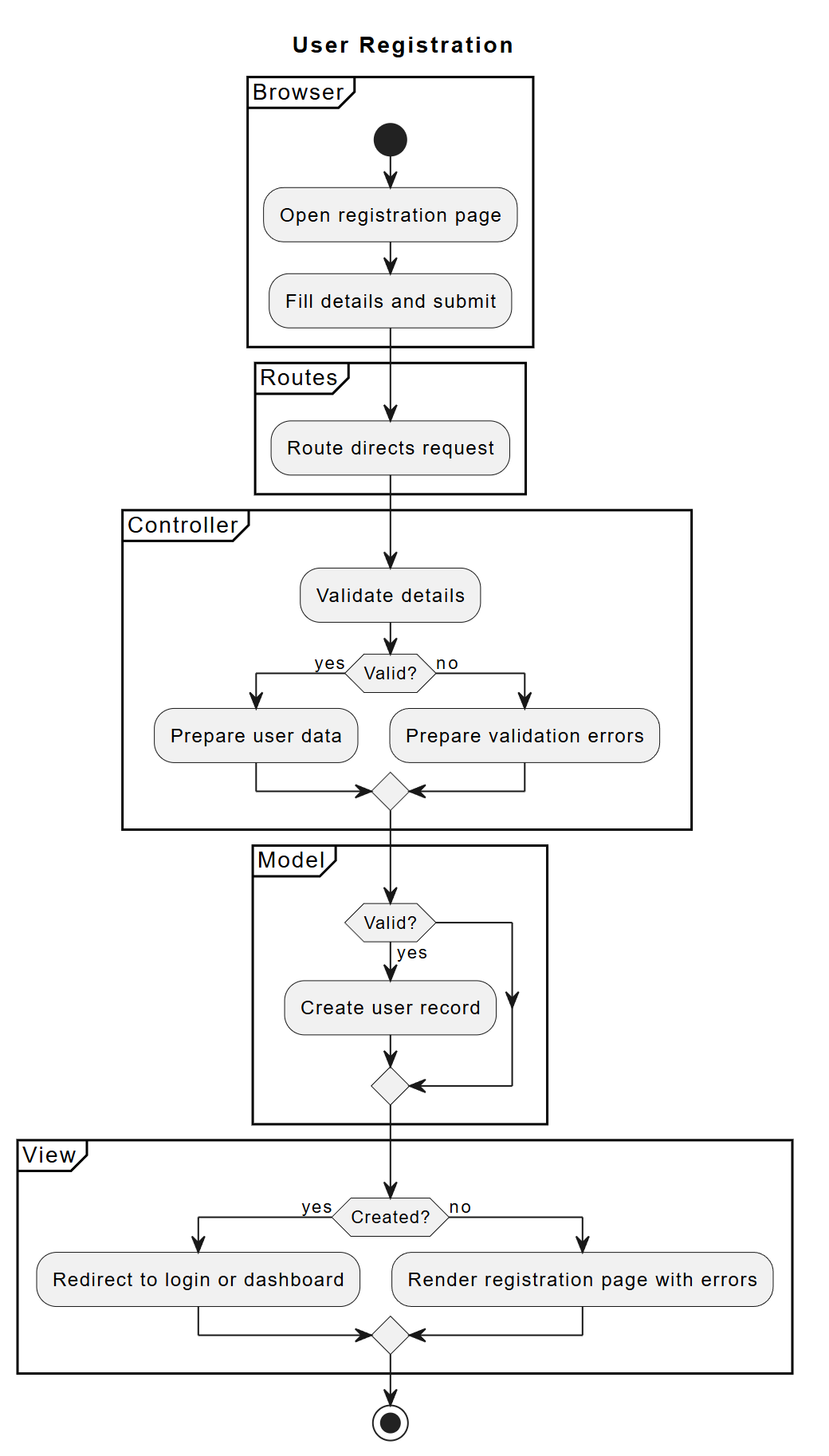
**Figure 3: Activity Diagram for Login**



**Figure 4: Activity Diagram for Admin Dashboard Control**

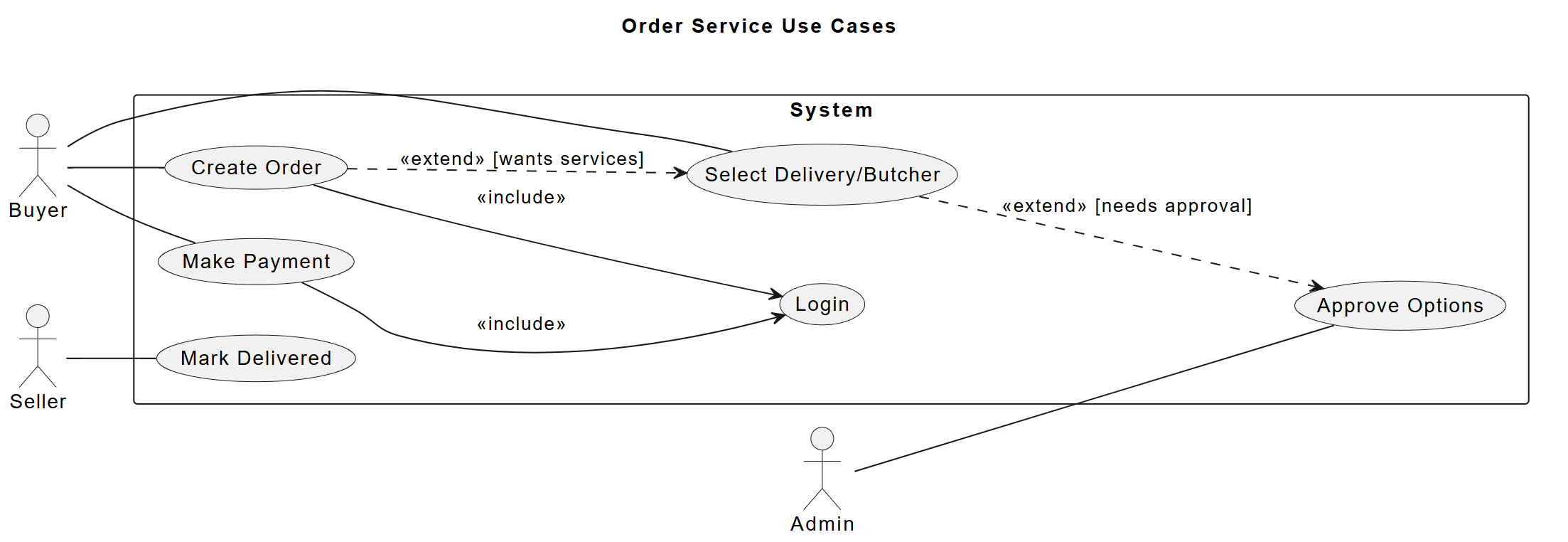


**Figure 5: Activity Diagram for Seller Verification**

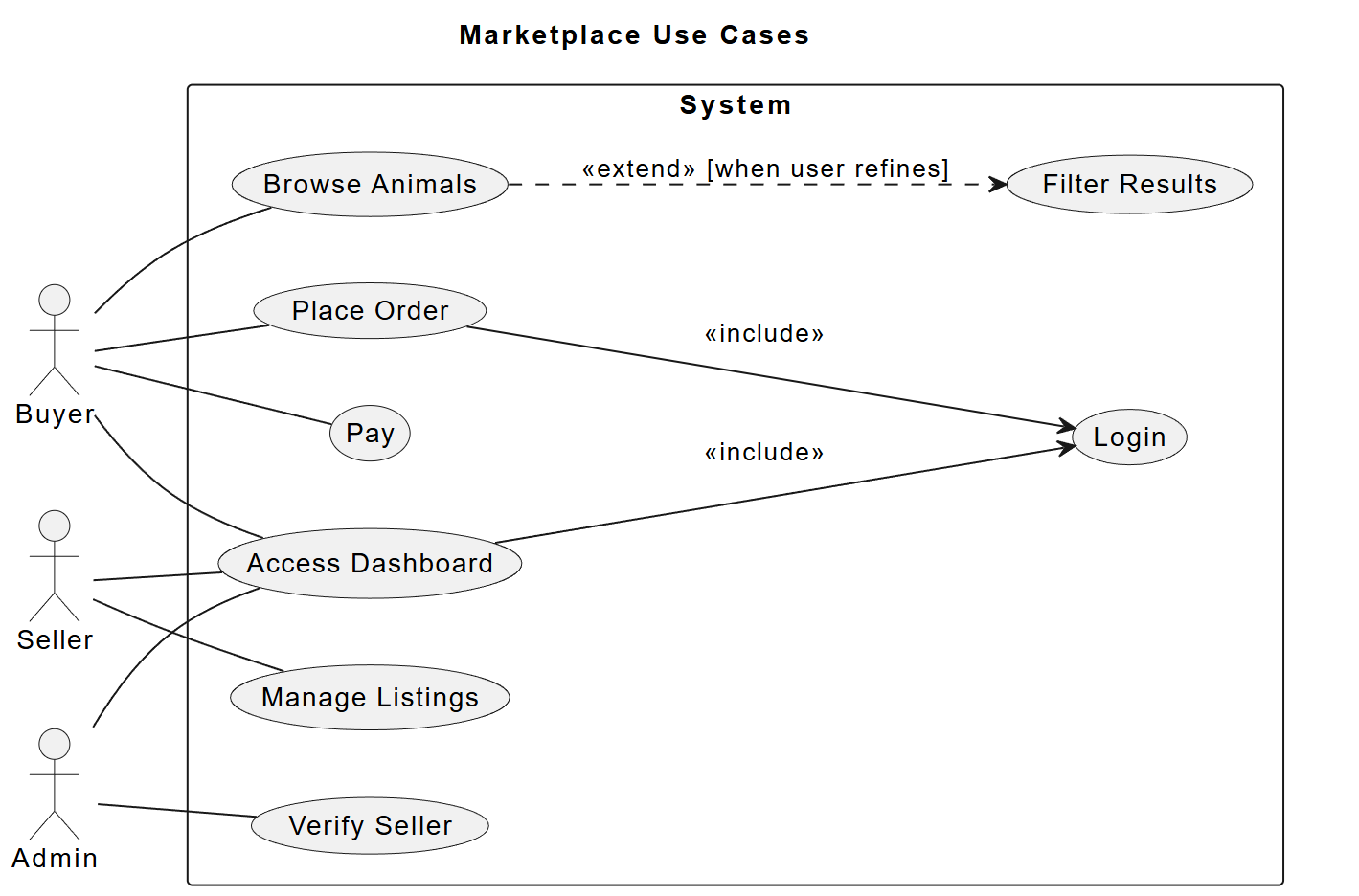


**Figure 6: Activity Diagram for Registration**

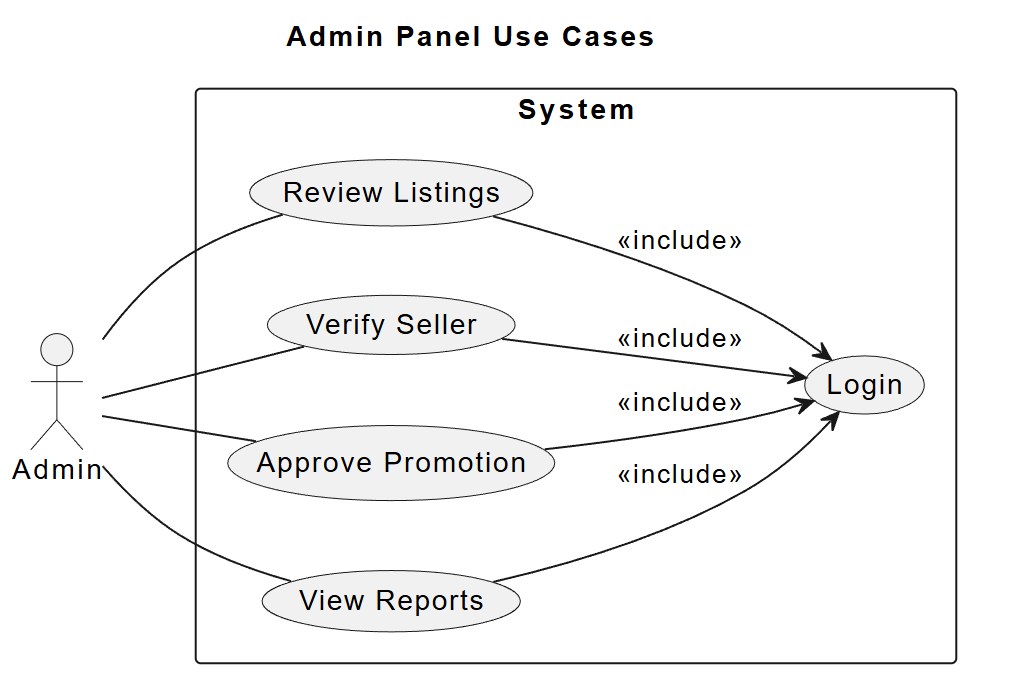
**Use Case diagrams** depict Buyer, Seller, and Admin with includes inside the system boundary (Login included by Dashboard access and Order placement) and extends for optional behaviors (Filter extends Browse Animals, Select Delivery/Butcher extends Create Order, Admin Approval extends Service Selection).



**Figure 7: Use Case Diagram for Ordering**

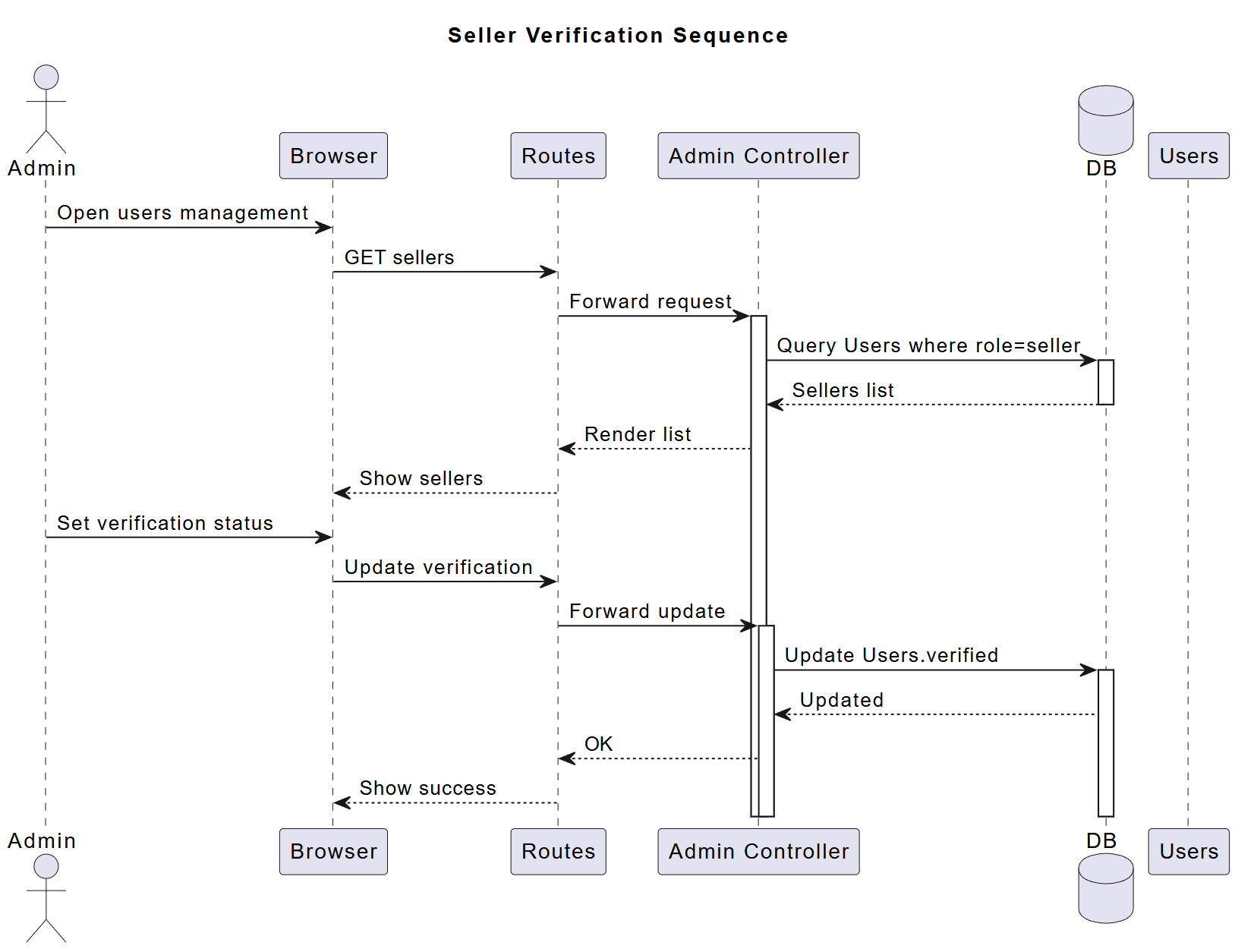


**Figure 8: Use Case Diagram for Dashboards**

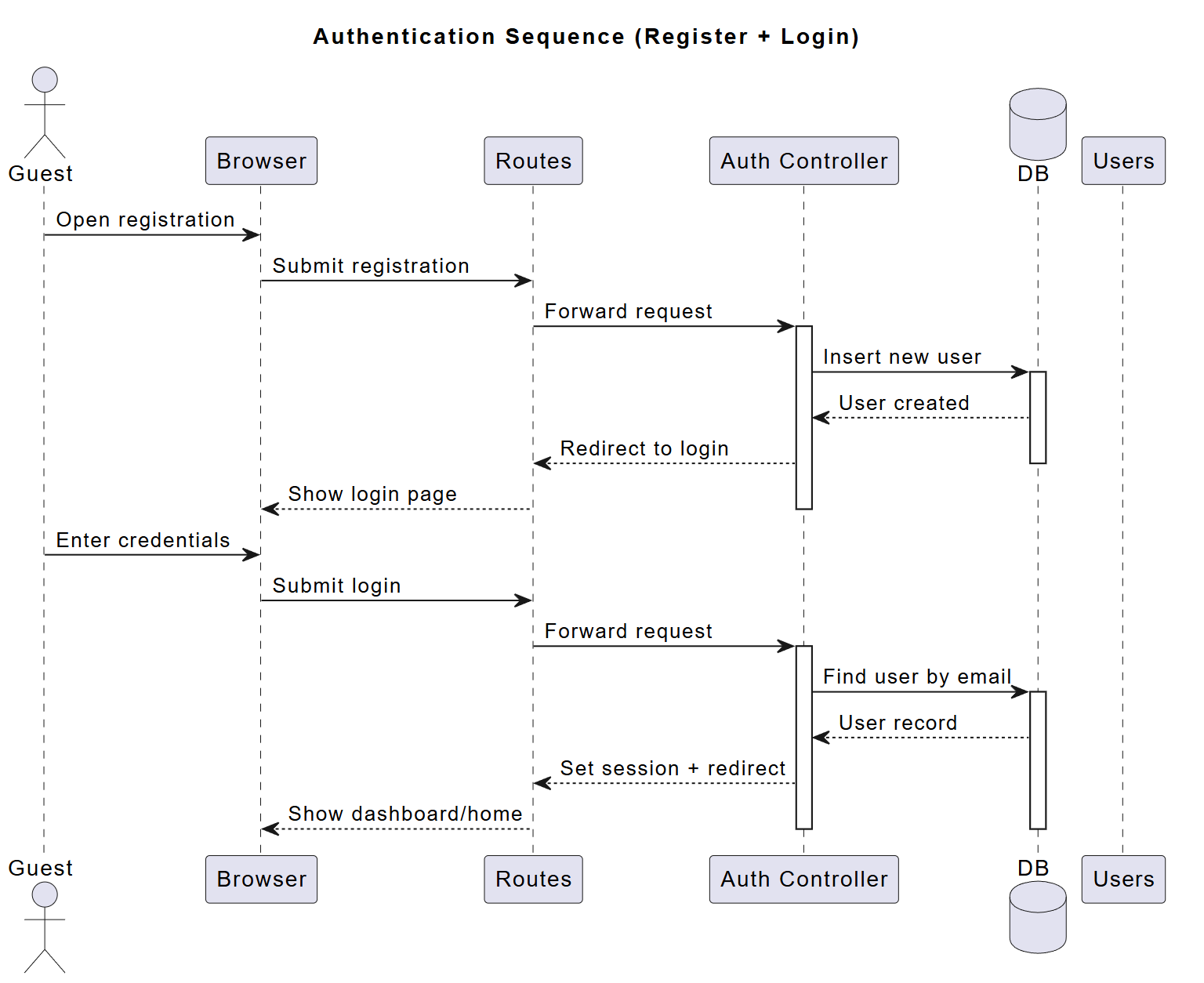


**Figure 9: Use Case Diagram for Admin control**

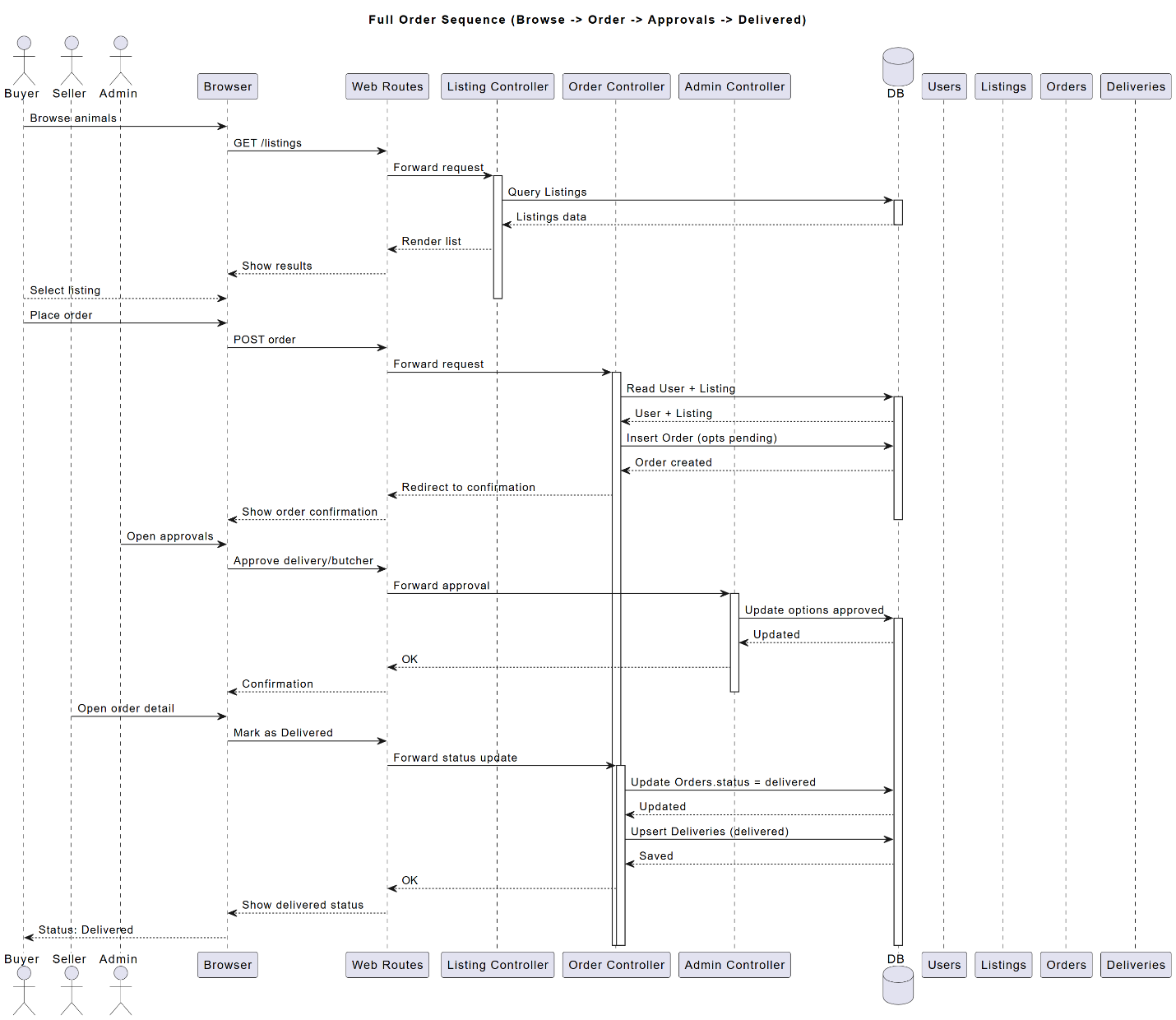
**Sequence diagrams** emphasize MVC and entity interactions: Full Order (Browser → Routes → Controllers → DB, with Users, Listings, Orders, Deliveries lifelines and activation periods); Seller Verification (Admin flow from dashboard to status change persisted in Users); and Auth (Registration followed by Login with session regeneration).



**Figure 10: Sequence Diagram for Seller verification**

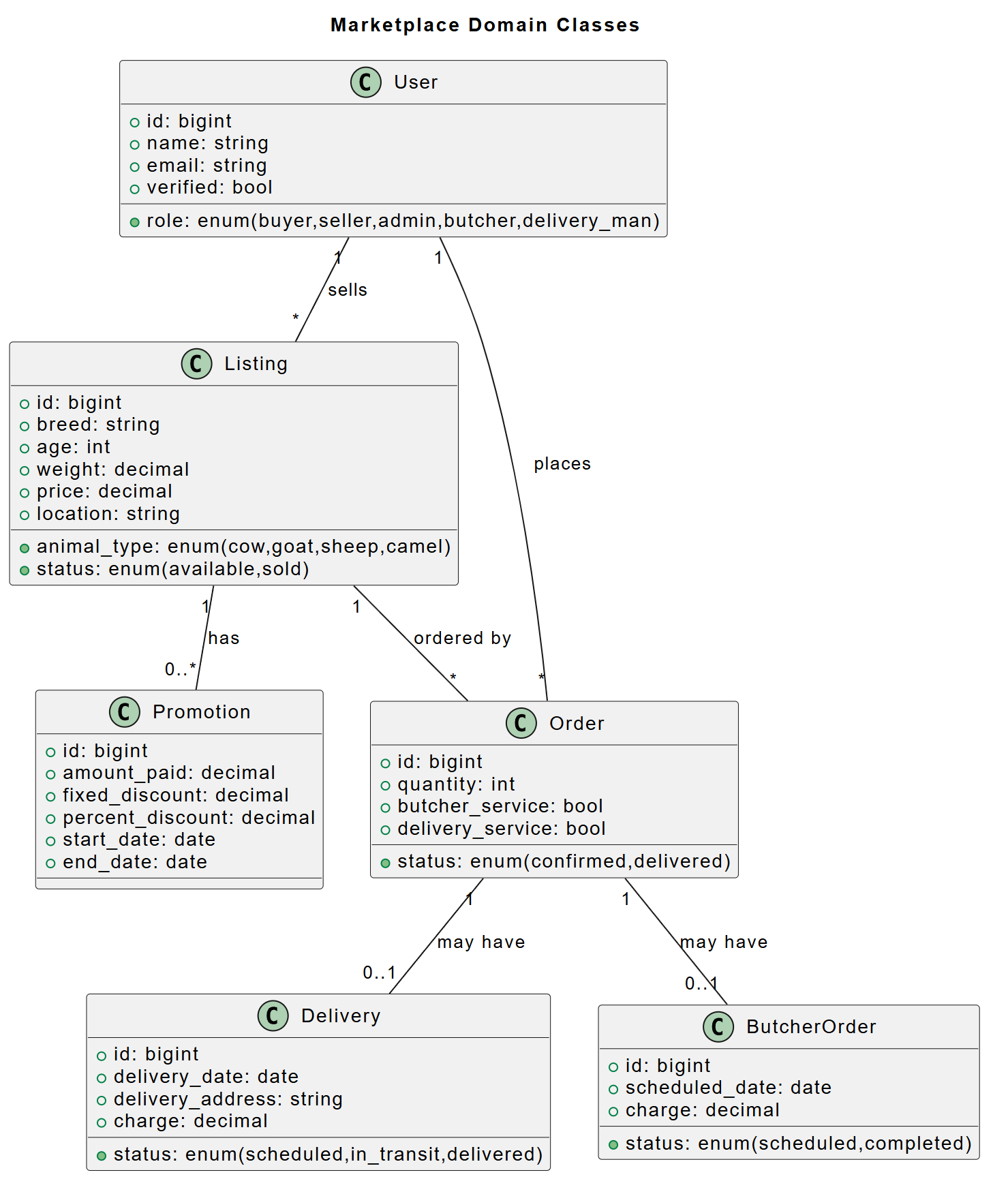


**Figure 11: Sequence Diagram for Authentication**



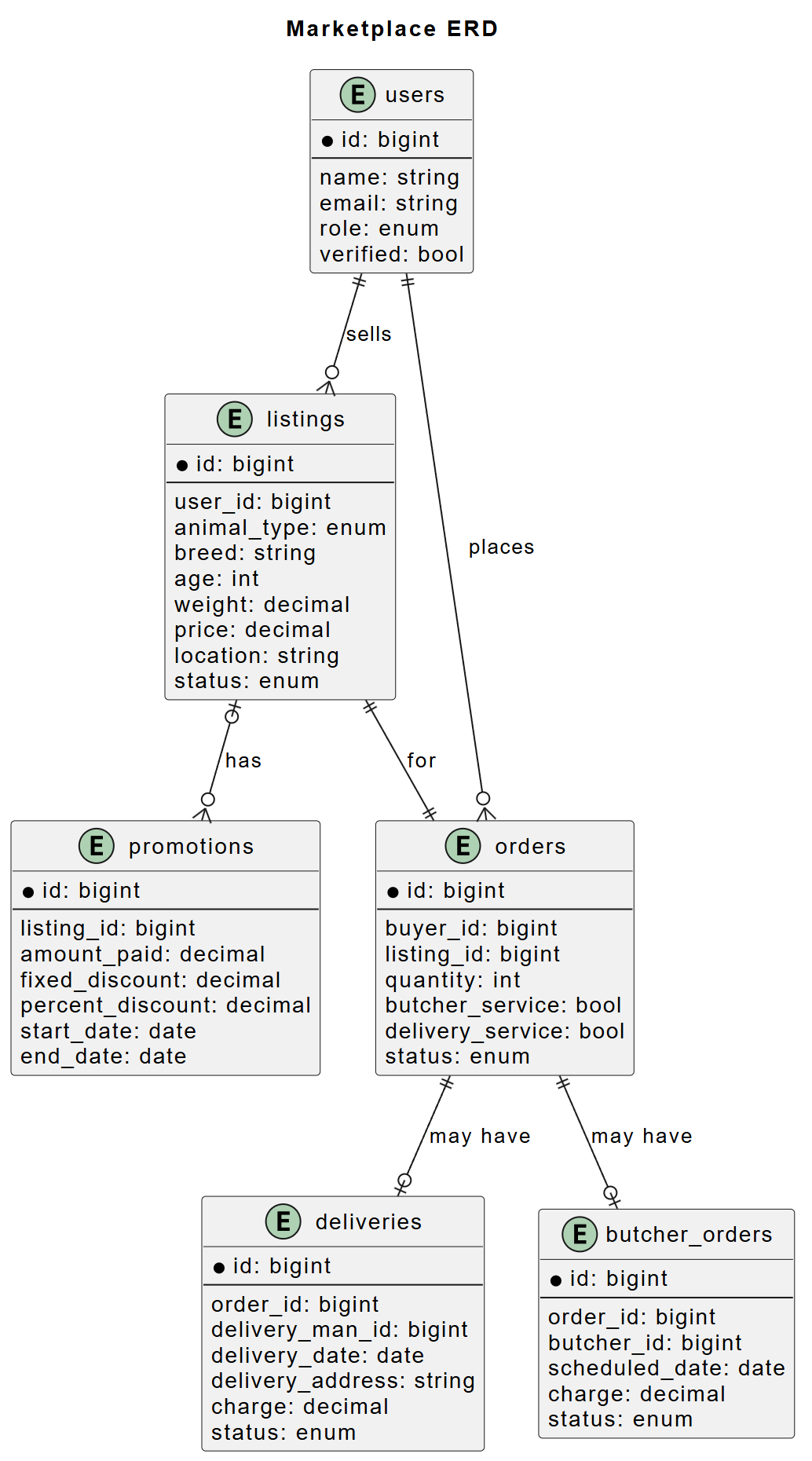
**Figure 12: Sequence Diagram for Ordering Life Cycle**

**Class diagram** defines domain classes, attributes, multiplicities, and generalizations where applicable.



**Figure 13: Class Diagram for YogeshMela**

**ERD** maps tables, keys, attributes, and cardinalities. The DFD (Gane & Sarson) shows Buyer and Payment Provider, processes for Place Order, Process Payment, and Fulfill Order, and data stores for Orders and Payments with directional data flows.

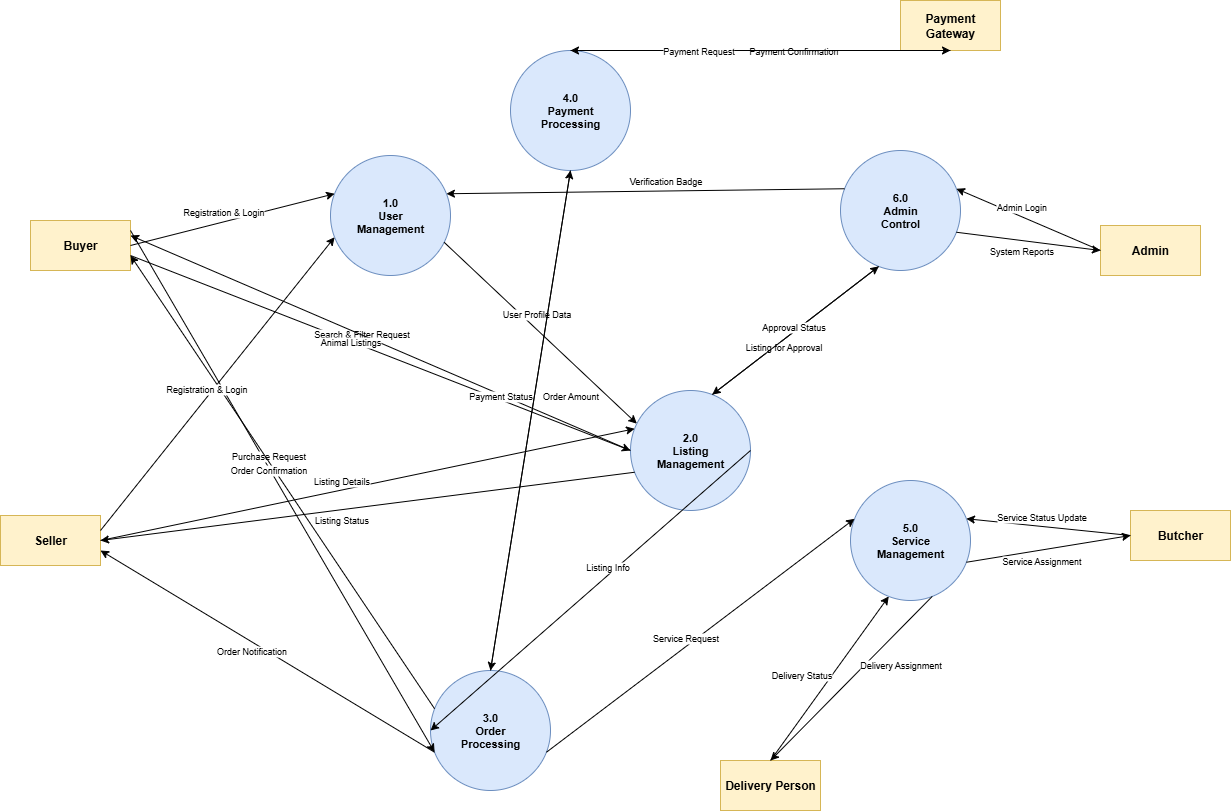


**Figure 14: ERD for YogeshMela**

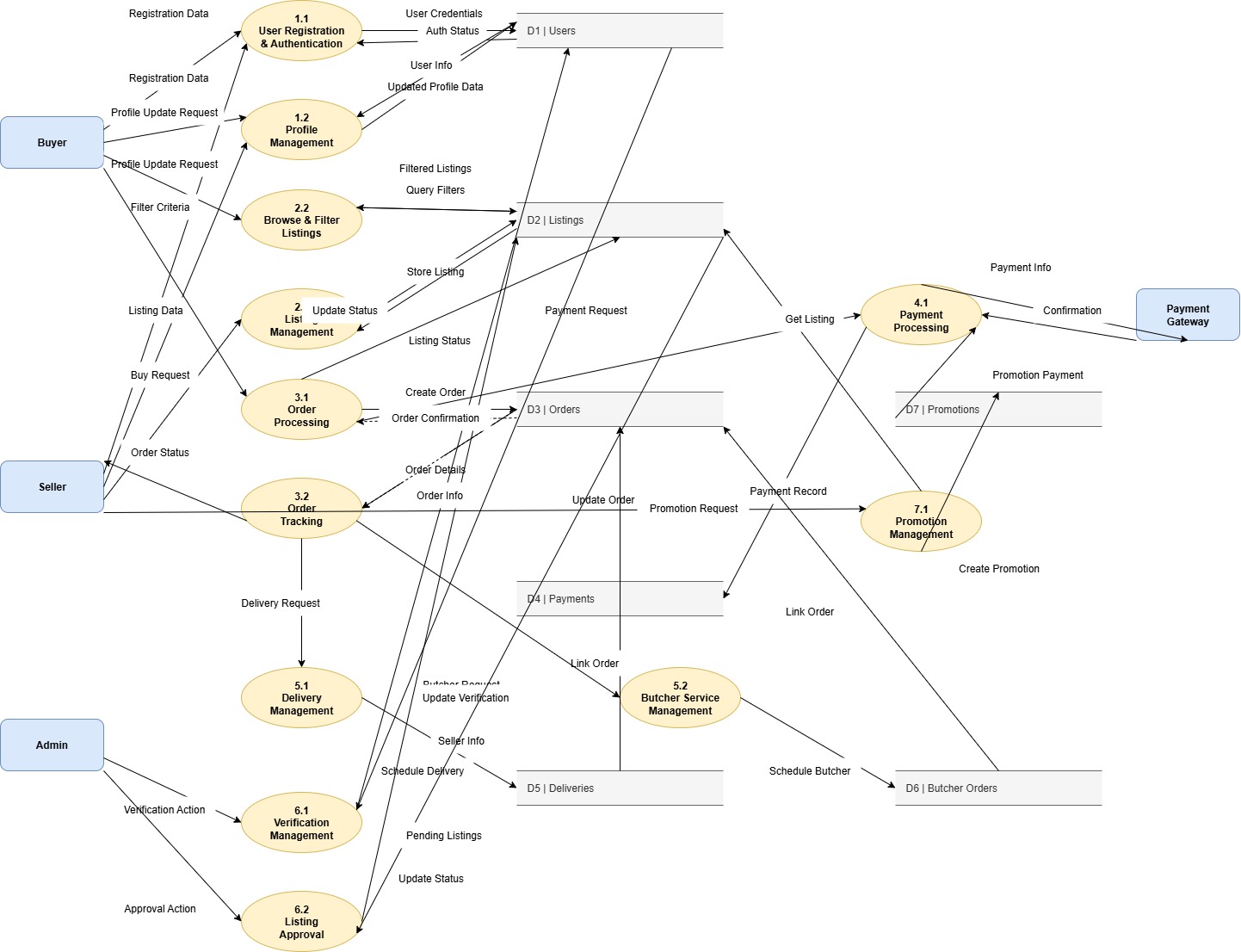
**DFD** shows how data moves through YogeshMela. The context level depicts the system as a single process interacting with Buyers, Sellers, Admin, and Payment Provider. Level 0 breaks it into major processes like Authentication, Listings, Orders, and Admin Oversight. Level 1 details subprocesses and database interactions.



**Figure 15: Context level DFD**

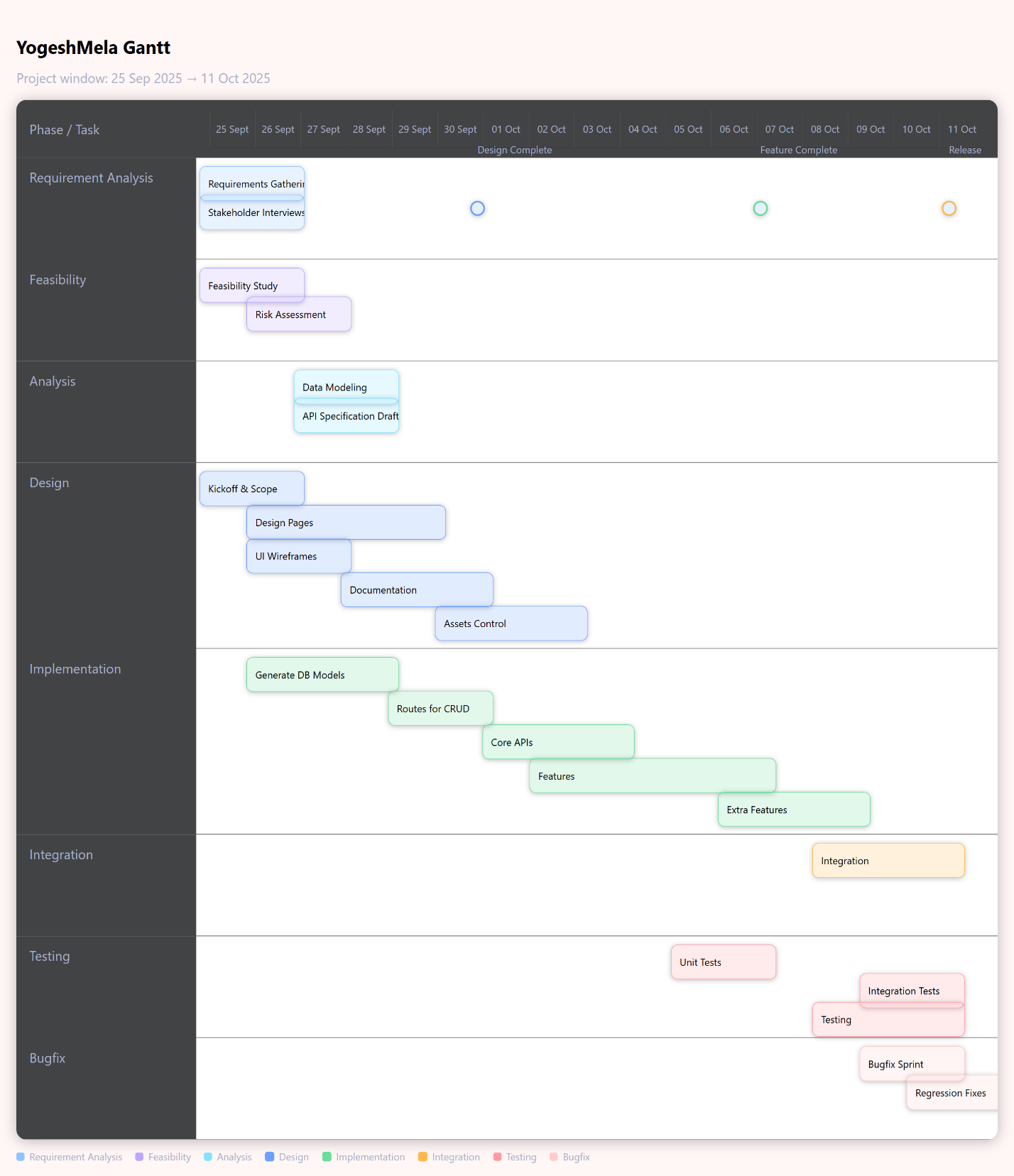


**Figure 16: Level-0 DFD**



**Figure 17: Level-1 DFD**

**Gantt chart** visualizes YogeshMela’s development timeline across SDLC phases: Requirement Analysis, Feasibility, Analysis, Design, Implementation, Integration, Testing, and Bugfix. It maps tasks, durations, dependencies, and milestones (Design Complete, Feature Complete, Release), helping track progress, allocate resources, and ensure timely delivery.



**Figure 18: Gantt Chart for YogeshMela**

**Discussion**

The development of **YogeshMela** followed a structured SDLC approach, emphasizing modularity, maintainability, and rapid iteration. Laravel was chosen as the backend framework due to its elegant MVC architecture, expressive routing, and Eloquent ORM, which facilitated quick database modeling, migrations, and seeding. The use of Blade templates allowed server-rendered pages that are fast, SEO-friendly, and accessible, while still providing a responsive, interactive interface for buyers, sellers, and admins.

The team adopted a phased approach: Requirement Analysis and Feasibility guided core functionality decisions, ensuring the marketplace covered essential workflows such as listing management, order placement, optional delivery and butcher services, and administrative moderation. Analysis and Design phases resulted in a normalized data model with clear relationships among Users, Listings, Orders, and associated services. Domain diagrams, sequence flows, and ERD diagrams were iteratively developed to ensure alignment between the intended features and implementation.

During Implementation, factories and seeders accelerated testing with realistic, reproducible data. Controllers were built to encapsulate business logic cleanly, while views provided intuitive interfaces for all user roles. Integration focused on linking listings, orders, services, promotions, and dashboards seamlessly, ensuring that cross-feature dependencies functioned correctly. Continuous testing was performed, including unit tests, integration tests, and validation of edge cases such as unauthorized access and invalid state transitions.

The team carefully considered trade-offs: a server-rendered approach was chosen over a SPA to simplify security, maintainability, and accessibility. Features such as role-based dashboards, optional service attachments, and promotion approval flows were developed to balance flexibility with operational control. Milestones and Gantt planning helped maintain timeline discipline, while iterative bug fixing and testing ensured a robust release candidate.

Performance optimizations were applied where necessary, including indexing foreign keys, eager-loading relationships to minimize database queries, and paginating listing results to handle scalability. Documentation, including ERD, DFD, activity, and sequence diagrams, was maintained in parallel with development to facilitate onboarding, future maintenance, and review processes.

**Conclusion**

**YogeshMela** successfully delivers a functional and extensible online livestock marketplace. The project demonstrates effective use of Laravel MVC principles, with clearly separated models, controllers, and Blade views. Core workflows—including listing management, order placement, optional delivery and butcher services, seller verification, promotions, and role-based dashboards—were implemented with strong focus on security, maintainability, and usability.

The development process emphasized reproducible data setups via seeders, structured testing, and modular, readable code to support future feature expansion. Integration of cross-functional components was carefully orchestrated to ensure consistency across workflows and role-specific interactions.

By leveraging server-rendered templates and relational database best practices, the system achieves a balance between speed, accessibility, and scalability. The project establishes a solid foundation for further enhancements such as payment integration, notification systems, analytics, and richer operational tools, while maintaining clarity in architecture and documentation. YogeshMela thus represents a reliable, maintainable, and user-centered solution for the online Qurbani marketplace.

**Software Specification:**

* **Framework:** Laravel (PHP 10.x) — MVC architecture for structured web development
* **Language:** PHP (Backend), HTML, CSS, JavaScript (Frontend)
* **Database:** MySQL (via Laravel Herd)
* **Local development environment:** HERD, XAMPP
* **Web Server:** Laravel Herd (built-in PHP & MySQL environment)
* **Frontend Tools:** Blade Templates, Bootstrap
* **Version Control:** Git & GitHub
* **IDE / Editor:** Visual Studio Code
* **Operating System:** Windows 11

**Github Repo Link**: <https://github.com/MishtiAloo/YogeshMela>

**Jira Scrum Project Link:** <https://ftahmid123.atlassian.net/jira/software/projects/YOG/summary>

**UI Mock-Up Design Link:** <https://app.visily.ai/projects/d83b24f8-18ae-474c-bba5-f33e848a5ea2/boards/2256589>

**References**

* <https://plantuml.com/>
* <https://app.diagrams.net/>
* <https://www.visual-paradigm.com/>
* <https://laravel.com/docs/12.x/>
* <https://app.visily.ai/projects?type=team>

**Team Members:**

* Dip Shekhor Datta (2107047)
* Megha Tania (2107057)
* Farhan Tahmid (2107059)