PART 1: Overview & Software Requirements Specification

1) Introduction:

a) Purpose

The primary purpose of the On-Road Vehicle Breakdown Assistance System is to:

Enable real-time roadside assistance: Use GPS and mapping APIs to locate and dispatch the closest qualified mechanic to a driver's broken-down vehicle, cutting response times in remote or underserved areas.

Streamline service ordering: Provide an intuitive interface for drivers to request services such as towing, tire changes, and fuel delivery without traditional call-center delays.

Enhance transparency and efficiency: Offer status updates, estimated time of arrival (ETA), and pricing information to drivers and mechanics to build trust and improve operational workflows.

The Software Requirements Specification (SRS) document is designed to:

1. Define clear requirements: Articulate exactly what the system must do (functional requirements) and how well it must perform (non-functional requirements), reducing ambiguity.

2. Align stakeholder expectations: Serve as a formal agreement between clients, mechanics, admins, and the development team on system scope, features, and constraints.

3. Guide development and testing: Provide a roadmap for architects, developers, and QA analysts to design, implement, and validate the system against documented standards.

4. Mitigate project risks: By detailing requirements upfront, identify potential issues early, enable accurate cost and schedule estimates, and minimize costly changes during later development phases.

b) Project Scope

The scope of the system covers three user-facing modules—Client, Mechanic, and Admin—plus shared Authentication Services, enabling

clients to locate mechanics, place and manage service orders, submit feedback, and share real-time location; mechanics to view and fulfill orders, offer prices, and update availability; and admins to manage user accounts, approve or block mechanics, and generate system reports. The SRS explicitly excludes payment processing, supply chain management, and advanced analytics beyond basic performance reporting.

c) Glossary and Abbreviations

Definitions for technical and non-technical terms.

Client: A user who requests on-road vehicle repair services.
Mechanic: A certified service provider who fulfills client orders.
Admin: System operator with privileges to manage users and system data.
ETA (Estimated Time of Arrival): Predicted time when a mechanic reaches the client's location.
2FA (Two-Factor Authentication): Security mechanism requiring two forms of identity verification.
MoSCoW: Prioritization scheme dividing requirements into Must, Should, Could, and Won't categories.
SRS: Software Requirements Specification document that details system requirements.

d) List of the System Stakeholders.

Clients: End users who request and track services.
Mechanics: Service providers who respond to orders and update service status.
Admins: Internal staff who manage user accounts, mechanics, and system configurations.
Developers: Engineers implementing the system components.
Testers/QA Analysts: Personnel validating system functionality against requirements.
Business Analysts/Product Owners: Individuals who define business needs and approve require

2) Functional Requirements:

User requirements	System requirements
Users should be able to register and log in to their accounts.	The system shall provide registration and authentication mechanisms for clients.
Users should be able to search for nearby mechanics based on their location.	The system shall allow clients to search and display mechanics filtered by proximity.
Users should be able to request assistance for vehicle breakdowns.	The system shall enable clients to create and submit service requests to mechanics.
Users should be able to view and manage their past service orders.	The system shall provide clients with access to their service history and order details.
Users should be able to provide feedback and rate mechanics after service completion.	The system shall allow clients to submit ratings and reviews for mechanics.
Users should be able to update their personal information, such as password and contact details.	The system shall provide functionality for clients to edit and update their profile information.
Users should be able to cancel or delete a service order if needed.	The system shall allow clients to cancel or delete pending service requests.
Users should be able to share their current location with the system for better service.	The system shall capture and utilize the client's real-time location data when shared.
Users should be able to search for mechanics by the nearest location.	The system shall sort and display mechanics based on proximity to the client's location.
Users should be able to view	The system shall provide detailed

detailed information about mechanics and get directions.	mechanic profiles and integrate with mapping services for directions.
Users should be able to request on-spot pickup if the vehicle cannot move.	The system shall allow clients to request immediate pickup services from available mechanics.
Users should be able to view the bill for the service provided.	The system shall generate and display billing information for each completed service.
Mechanics should be able to register and log in to their accounts.	The system shall provide registration and authentication mechanisms for mechanics.
Mechanics should be able to view assigned service orders.	The system shall display a list of service requests assigned to the mechanic.
Mechanics should be able to provide solutions and update the status of service orders.	The system shall allow mechanics to input solutions and update order statuses accordingly.
Mechanics should be able to view feedback and ratings from clients.	The system shall display client feedback and ratings to the respective mechanic.
Mechanics should be able to offer a price estimate for a service request.	The system shall enable mechanics to submit price estimates for client approval.
Mechanics should be able to update their password.	The system shall provide functionality for mechanics to change their account passwords.
Mechanics should be able to update their availability status.	The system shall allow mechanics to set and update their current availability.
Mechanics should be able to accept or decline incoming service requests.	The system shall provide options for mechanics to accept or reject new service assignments.

Mechanics should be able to view their service history.	The system shall provide access to a log of past services completed by the mechanic.
Mechanics should be able to communicate with clients through in-app chat.	The system shall facilitate real-time messaging between mechanics and clients.
Admins should be able to log in securely.	The system shall provide authentication mechanisms for administrators.
Admins should be able to view information about clients and mechanics.	The system shall display detailed profiles and data for all users.
Admins should be able to search for mechanics and clients based on location.	The system shall provide search functionality filtered by geographic location.
Admins should be able to update mechanic information, such as status.	The system shall allow admins to edit mechanic profiles and update their status.
Admins should be able to add new mechanics to the system.	The system shall provide functionality to register and onboard new mechanics.
Admins should be able to delete mechanics from the system.	The system shall allow admins to remove mechanic accounts as needed.
Admins should be able to update their own passwords.	The system shall provide functionality for admins to change their account passwords.
Admins should be able to generate performance reports for mechanics and clients.	The system shall compile and present performance metrics and reports.
Admins should be able to send system-wide notifications to users.	The system shall enable admins to broadcast messages to all users.

Admins should be able to handle payment disputes between clients and mechanics.	The system shall provide tools for admins to review and resolve payment issues.
Admins should be able to archive old service orders for record-keeping.	The system shall allow archiving of completed or outdated service orders.
The system should support two-factor authentication for enhanced security.	The system shall implement two-factor authentication mechanisms during login.

Requirements' Priorities

Category	Priority Description
Must Have	User authentication; search mechanics by location; place/cancel orders; mechanic notifications.
Should Have	Feedback submission; update personal profile; request on-spot pickup; view directions.
Could Have	Search by nearest mechanic; share live GPS location; in-app chat; basic analytics dashboards.
Won't Have	Supply chain management; advanced AI-based mechanic recommendations (future phase).

3) Non-functional Requirements:

(1) **Performance:** The system should provide fast and efficient responses to user interactions, ensuring smooth operation during normal use.
(2) **Availability:** it should be accessible at all times, allowing users to request and provide assistance whenever needed.
(3) **Scalability**: The system should be capable of handling growth in the number of users, mechanics, and service requests without affecting performance.
(4) **Security:** User data and communications should be protected through encryption and secure authentication methods, with appropriate access control for different roles.
(5) **Usability:** The interface should be user-friendly and intuitive, enabling users to navigate and use the application easily without requiring prior training.
(6) **Maintainability:** The system should be designed in a modular and well-documented way, allowing developers to update and maintain the application efficiently.
(7) **Reliability:**it should perform consistently and remain stable, even in the case of unexpected errors or system stress.
(8) **Portability:**it should work across multiple devices and platforms, including web and mobile environments, with consistent behavior.
(9) **Backup and Recovery:**The system should include a mechanism to regularly back up critical data and recover from data loss or system failure effectively.
(10) **Compliance:**it should adhere to applicable laws and standards related to data protection and user privacy.

4) Design & Implementation Constraints.

- The system must be built using PHP for backend and HTML, Bootstrap for frontend to align with existing infrastructure.
- The application must support major modern browsers (Chrome, Firefox, Safari) and mobile WebView implementations on iOS and Android.

5) System Evolution:

a) Anticipated changes:

- Addition of payment processing and invoicing modules.

- Integration with real-time chat or video support for remote diagnostics.

b) Design accommodation:

- Employ a modular microservices architecture to allow independent deployment and scaling of new features (e.g., payment microservice).
- Use well-defined RESTful APIs and message queues to decouple components, facilitating future integrations with third-party services.
- Adopt schema-versioning and backward-compatible database migrations to evolve the data model without downtime.

6) Requirements Discovery Approaches

Workshops & Focus Groups: Facilitate collaborative sessions with clients, mechanics, and admins to uncover requirements via brainstorming and prioritization exercises.

Interviews: Conduct structured interviews with key stakeholders to gather detailed use-case scenarios, acceptance criteria, and domain constraints.

Observations & Shadowing: Shadow mechanics in the field to understand real-world workflows and pain points for accurate requirement capture.

Prototyping: Develop low-fidelity UI mockups and interactive prototypes to validate user interactions (e.g., order placement flow) and refine requirements iteratively.

Document Analysis: Review existing support logs, SOPs, and mechanic manuals to derive functional and data requirements.

7) Requirements Validation Techniques

Requirements Inspections: Conduct peer reviews and walkthroughs of the SRS with stakeholders to detect ambiguities, inconsistencies, and omissions.

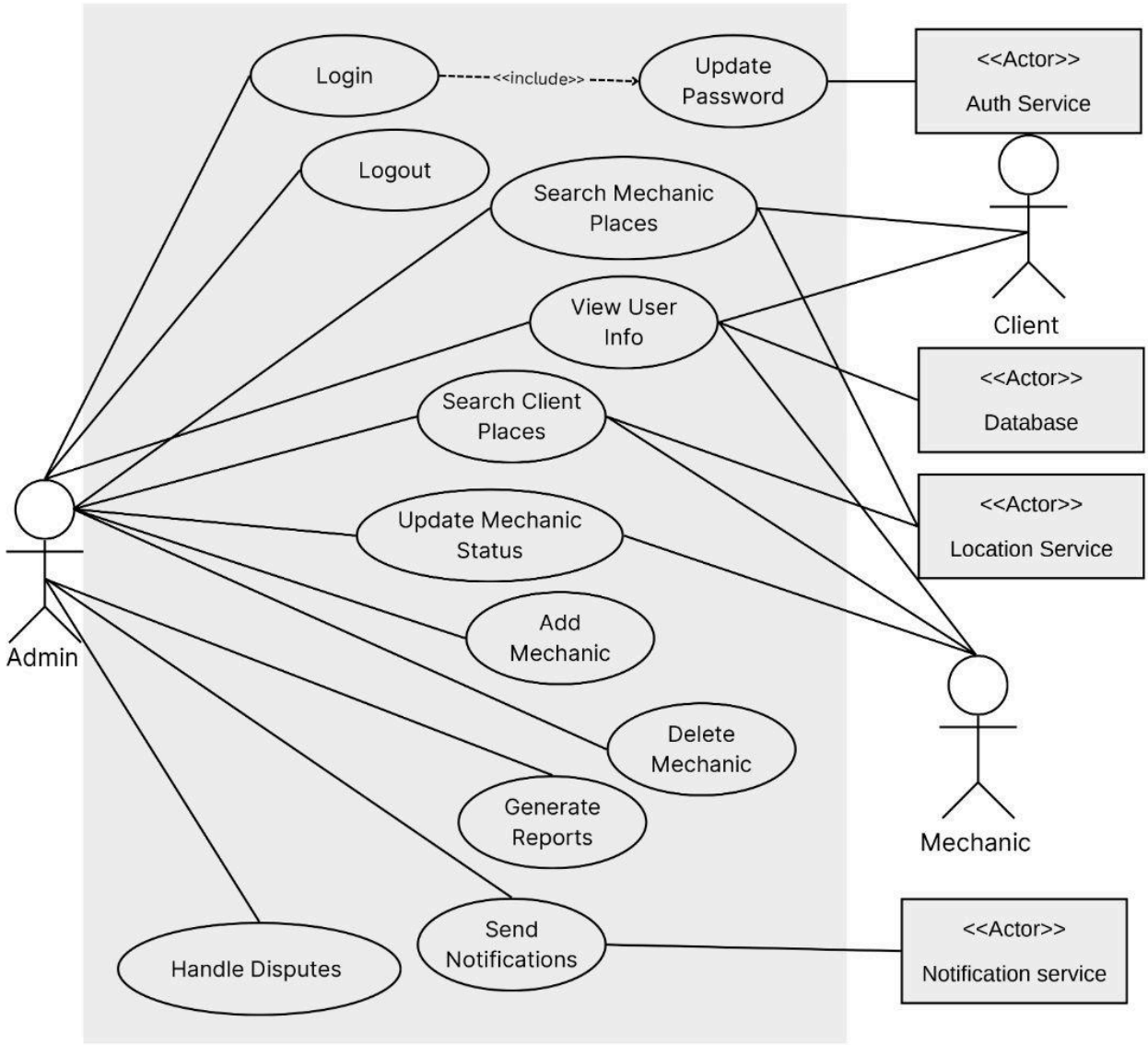
Prototyping & Usability Testing: Validate UI/UX requirements by testing prototypes with real users to ensure the system meets user expectations.

Test-Case Generation: Derive test cases directly from requirements to ensure each requirement is verifiable and testable.

Traceability Matrix: Maintain a requirements traceability matrix mapping requirements to design elements, code modules, and test cases for completeness verification.
Acceptance Testing Workshops: Conduct joint acceptance testing sessions with clients to confirm the implemented features meet the documented requirements.

System Design & Models

Admin use case diagram



Use Case Description
Login Description:

Field	Detail
Goal	Allow Admin to securely access the system.
Initiator	Admin
Preconditions	• Admin must be registered with valid credentials. • Authentication service must be available. • Device has a stable internet connection.
Postconditons	• Admin is authenticated and session is established. • Admin can access authorized system modules.
Main Success Scenario	• Admin navigates to login screen. • Enters correct username and password. • System validates credentials via auth service. • Successful login redirects admin to dashboard.
Alternative Scenario	• Incorrect credentials entered. • Account is locked after multiple failures. • Authentication service unavailable. • Network connection error during login.

Logout Description

Field	Detail
Goal	Allow admin to securely exit the system.
Initiator	Admin
Preconditions	• Admin is logged into the system. • Active session exists.
Postconditons	• Session is terminated. • Sensitive data is cleared from memory.
Main Success Scenario	• Admin clicks 'Logout'. • Session is closed and user is redirected to login page.
Alternative Scenario	• Logout fails due to session timeout or backend error.

Update Password Description

Field	Detail
Goal	Enable admin to securely change their password.
Initiator	Admin
Preconditions	• Admin is logged in. • Current password is known.
Postconditons	• Password is updated and policy-verified.
Main Success Scenario	• Admin enters old and new password. • System validates and updates credentials.
Alternative Scenario	• Password too weak. • Current password is incorrect.

View User Information Description

Field	Detail
Goal	Enable admin to view all registered clients and mechanics.
Initiator	Admin
Preconditions	• Admin is logged in with appropriate permissions. • User database is accessible.
Postconditons	• List of users (clients/mechanics) is displayed. • Data is fetched in real-time.
Main Success Scenario	• Admin selects 'View Users'. • System fetches data from user database. • Data displayed in organized view.
Alternative Scenario	• User database is down. • Query fails or times out.

Search For Mechanic Places Description

Field	Detail
Goal	Allow admin to locate mechanic distribution by location.
Initiator	Admin
Preconditions	• Admin is authenticated. • Location service and mechanic registry are functional.
Postconditons	• Map or list of mechanic locations is displayed.
Main Success Scenario	• Admin inputs location or filter. • System queries mechanic registry with location filter. • Results are shown on map or in list.
Alternative Scenario	• No mechanics found in specified area. • Location service unavailable.

Search For Client Places Description

Field	Detail
Goal	Allow admin to locate client distribution by location.
Initiator	Admin
Preconditions	• Admin is authenticated. • Client registry and location services are active.
Postconditons	• List or map of client locations is presented.
Main Success Scenario	• Admin specifies region or uses location filter. • System retrieves matching client addresses. • Data is presented visually.
Alternative Scenario	• No clients found in area. • Map service fails.

Update Mechanic Status Description

Field	Detail
Goal	Enable admin to change a mechanic's status (e.g., active, blocked).

Initiator	Admin
Preconditions	• Admin is logged in. • Target mechanic is already registered.
Postconditons	• Mechanic's status is updated in system.
Main Success Scenario	• Admin navigates to mechanic profile. • Changes status and submits. • System stores new status.
Alternative Scenario	• Mechanic not found. • Update fails due to server error.

Add Mechanic Description

Field	**Detail**
Goal	Let admin register a new mechanic into the system.
Initiator	Admin
Preconditions	• Admin has necessary access rights. • Mechanic info is valid and complete.
Postconditons	• Mechanic account is added and active. • Confirmation is sent.
Main Success Scenario	• Admin fills in mechanic form. • System validates and stores record. • Mechanic receives confirmation.
Alternative Scenario	• Form contains errors. • Database update fails.

Delete Mechanic Description

Field	**Detail**
Goal	Allow admin to permanently remove a mechanic from the system.
Initiator	Admin
Preconditions	• Admin is logged in. • Target mechanic exists in system.

Postconditons	• Mechanic record is removed. • Associated assignments are cleared.
Main Success Scenario	• Admin selects mechanic and confirms deletion. • System deletes mechanic from database.
Alternative Scenario	• Mechanic ID invalid or already removed. • Database error occurs.

Send System Notifications Description

Field	**Detail**
Goal	Allow admin to send notifications to mechanics or clients.
Initiator	Admin
Preconditions	• Admin has message content ready. • Notification service is active.
Postconditons	• Recipients receive messages via preferred channels.
Main Success Scenario	• Admin composes and sends message. • System queues and delivers message.
Alternative Scenario	• Delivery fails to some users. • Notification service down.

Handle Disputes Description

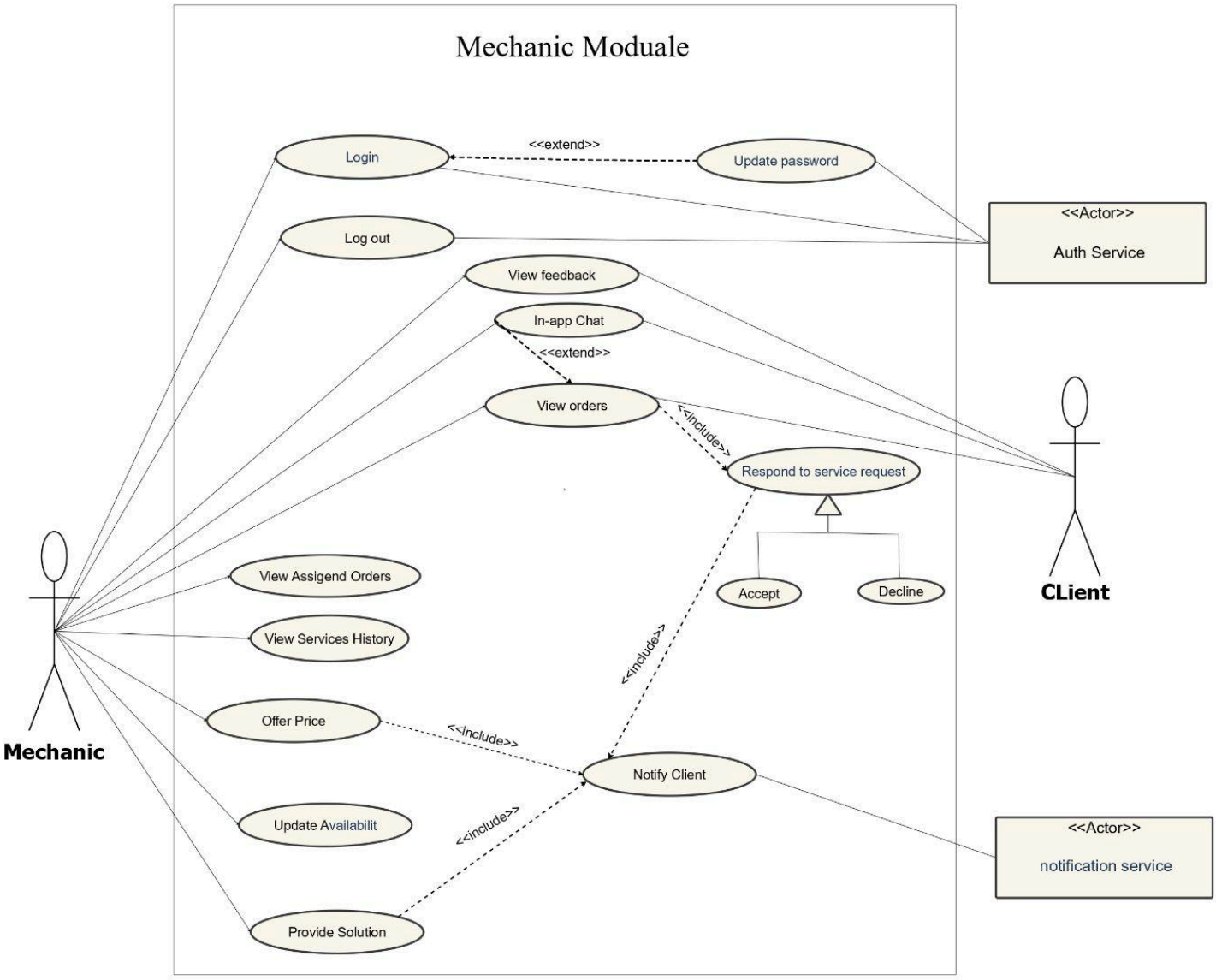
Field	**Detail**
Goal	Permit admin to resolve conflicts regarding transactions.
Initiator	Admin
Preconditions	• Dispute is flagged by client or mechanic. • Admin is logged in with dispute resolution rights.
Postconditons	• Dispute is resolved and logged. • Involved parties are notified.
Main Success Scenario	• Admin reviews dispute record. • Investigates transaction.

	• Confirms resolution and notifies users.
Alternative Scenario	• Insufficient data to resolve dispute. • Error updating transaction record.

Generate Reports Description

Field	**Detail**
Goal	Allow admin to create service performance analytics.
Initiator	Admin
Preconditions	•Admin is authenticated. •Database access is enabled.
Postconditons	•Report is generated and available for download/view.
Main Success Scenario	•Admin selects report type and period. •System compiles data and formats report.
Alternative Scenario	•Data missing for selected time. •Report generation fails.

Mechanic Use Case Diagram



Login Description

Field	Detail
Goal	Allow mechanic to securely access the system.
Initiator	Mechanic
Preconditions	Mechanic has a registered account. Mechanic has valid credentials (username and password). The authentication service is online and reachable. The device used has an active internet connection.
Postconditons	Mechanic is authenticated. A secure session is created. Mechanic is granted access to authorized parts of the system.
Main Success Scenario	Mechanic opens the app and navigates to the login screen. Enters valid username and password. System calls the authentication service. Credentials are validated successfully. Mechanic is redirected to the main dashboard.
Alternative Scenario	Incorrect username or password is entered. Account is locked after multiple failed attempts. Authentication service is down. Network connection failure during login request.

Logout Description

Field	Detail
Goal	Enable the mechanic to exit the system safely.
Initiator	Mechanic
Preconditions	Mechanic is logged into the system. Session is currently active.
Postconditons	Session is terminated. Any sensitive data is cleared from the local cache. Mechanic is returned to the login screen.
Main Success Scenario	Mechanic clicks the "Logout" button. The system ends the session.

	User is redirected to the login page.
Alternative Scenario	Session timeout occurs before logout is triggered. Logout fails due to server timeout. Mechanic mistakenly logs out during an active task.

Update Password Description

Field	**Detail**
Goal	Permit the mechanic to change their password.
Initiator	Mechanic
Preconditions	Mechanic is logged in. Mechanic knows the current password. Password meets security policy.
Postconditons	Password is securely updated in the system. Mechanic must use the new password on the next login. Session may optionally be refreshed.
Main Success Scenario	Mechanic navigates to profile settings. Enters current and new password. System validates current password and password strength. System updates the password.
Alternative Scenario	Current password is incorrect. New password does not meet security requirements. Server error during update. Password change fails due to connection timeout.

View Order Description

Field	**Detail**
Goal	Enable mechanic to browse available service requests.
Initiator	Mechanic
Preconditions	Mechanic is logged in. Mechanic has the correct permissions.
Postconditons	Mechanic sees a list of all available service requests. Orders are updated in real time if new ones come in.

Main Success Scenario	Mechanic accesses "Available Orders" tab. System retrieves all open service requests. Orders are displayed with details (e.g., location, issue).
Alternative Scenario	No available orders are present. Server fails to fetch the data. Interface loading failure or delay.

View Assigned Orders Description

Field	**Detail**
Goal	Allow mechanic to view orders assigned to them.
Initiator	Mechanic
Preconditions	Mechanic is logged in. One or more orders are assigned to the mechanic.
Postconditons	Mechanic sees all currently assigned service orders.
Main Success Scenario	Mechanic opens the “Assigned Orders” tab. System queries the database for orders linked to their ID. Orders are displayed along with deadlines and status.
Alternative Scenario	No orders assigned to the mechanic. Orders are outdated or already completed. Data retrieval failure.

Respond to Service Requests Description

Field	**Detail**
Goal	Let mechanic accept or decline a service job.
Initiator	Mechanic
Preconditions	Mechanic is logged in. Order is not expired or already assigned. Mechanic has permission to respond.
Postconditons	Decision (Accept/Decline) is saved. Client is notified of the mechanic’s response.
Main Success Scenario	Mechanic selects a request. Chooses to accept or decline.

	System records the decision. Notification service alerts the client.
Alternative Scenario	Service request expired or assigned to someone else. Notification service fails. Response action fails to save due to backend issue.

Accept/Decline Description

Field	**Detail**
Goal	Update the request status based on mechanic's decision.
Initiator	Mechanic
Preconditions	Service request is valid and pending.
Postconditons	Status is updated in the system. Client is notified via notification service.
Main Success Scenario	Mechanic selects "Accept" or "Decline". System stores the decision. Notification is sent to the client.
Alternative Scenario	Request already taken by another mechanic. Mechanic clicks by mistake. System error updating the request.

Provide Solution Description

Field	**Detail**
Goal	Allow mechanic to send a technical solution to the client.
Initiator	Mechanic
Preconditions	Mechanic is assigned to the request. Request is in “solution pending” state.
Postconditons	Solution is saved. Client is notified of the solution provided.
Main Success Scenario	Mechanic types or uploads a solution. System stores the solution and logs it. Notification service alerts the client.

Alternative Scenario	Mechanic provides incomplete or unsupported format. Storage fails. Notification does not reach the client.

Offer Price Description

Field	**Detail**
Goal	Enable the mechanic to propose a service cost.
Initiator	Mechanic
Preconditions	Mechanic assessed the issue. Mechanic is authenticated.
Postconditons	Price is recorded and sent to client. Client receives notification.
Main Success Scenario	Mechanic enters price. System saves the offer. Notification is sent to client
Alternative Scenario	Price not accepted due to invalid format. Notification fails. System does not register the price due to network issues.

Update Availability Description

Field	**Detail**
Goal	Let mechanic show availability to clients.
Initiator	Mechanic
Preconditions	Mechanic is logged in.
Postconditons	Status is updated (Available/Busy).
Main Success Scenario	Mechanic toggles availability. System updates the record.
Alternative Scenario	Server fails to save the update. Mechanic unintentionally switches status. Interface delay misrepresents actual state.

View Service History Description

Field	Detail
Goal	Permit mechanic to view completed past services.
Initiator	Mechanic
Preconditions	Mechanic is logged in. Mechanic has completed orders.
Postconditons	All completed services are shown.
Main Success Scenario	Mechanic selects history tab. System fetches past data. Services are listed chronologically.
Alternative Scenario	No completed services available. Data fetch fails. History list shows incorrect entries.

View Feedback Description

Field	Detail
Goal	Allow mechanic to read client ratings and comments.
Initiator	Mechanic
Preconditions	Client has submitted feedback. Feedback module is active..
Postconditons	Mechanic can view comments, ratings, and client suggestions.
Main Success Scenario	Mechanic opens feedback section. System displays the reviews.
Alternative Scenario	No feedback exists. Display issue or feedback not loading. Ratings are outdated or mismatched.

In-App Chat Description

Field	**Detail**
Goal	Enable real-time communication between mechanic and client.
Initiator	Mechanic or Client
Preconditions	Active service request exists. Both users are online.
Postconditons	Messages are exchanged through chat module.
Main Success Scenario	Chat window is opened. Messages are sent and received. Chat history is stored.
Alternative Scenario	One party is offline. Chat server is down. Delays in message delivery.

NotifyClient Description

Field	**Detail**
Goal	Send system-generated updates to the client..
Initiator	System (triggered by other use cases)
Preconditions	Trigger event occurs (e.g., solution sent, price offered).
Postconditons	Client receives a system-generated notification.
Main Success Scenario	System detects trigger. Notification is sent via SMS, app, or email. Client receives update in real time.
Alternative Scenario	Client’s device is offline. Notification service is temporarily unavailable.

Share Location Description

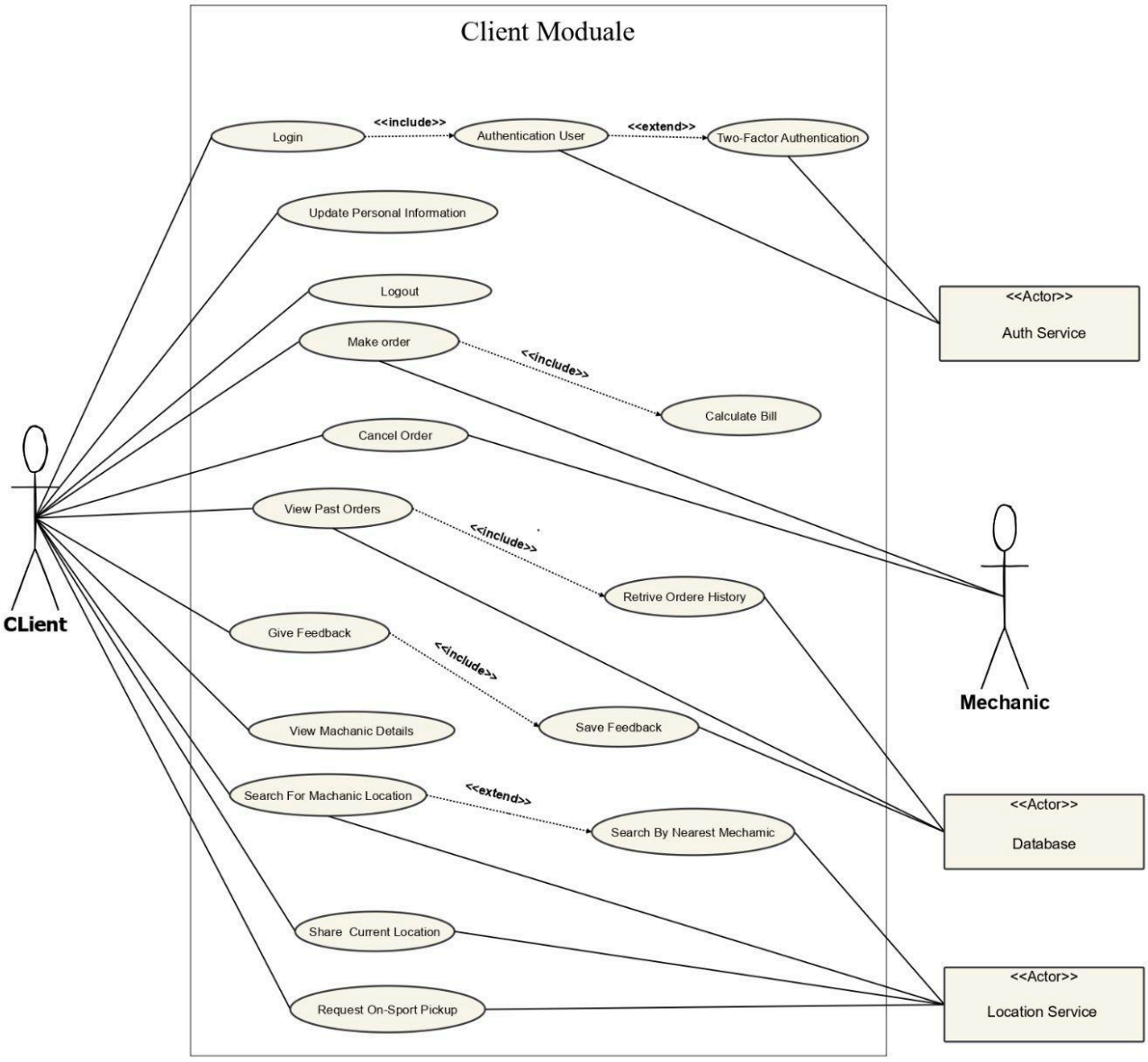
Field	**Detail**
Goal	Allow both parties to exchange their real-time locations.

Initiator	Mechanic or Client
Preconditions	Both users are online. Location sharing permissions are granted.
Postconditons	Locations are displayed via embedded map.
Main Success Scenario	Mechanic enables “Share Location”. Client receives real-time map view. Location is updated continuously..
Alternative Scenario	GPS disabled on device. One party revokes location access. Network error prevents location update.

View Service History Description

Field	**Detail**
Goal	Permit mechanic to view completed past services.
Initiator	Mechanic
Preconditions	Mechanic is logged in. Mechanic has completed orders.
Postconditons	All completed services are shown.
Main Success Scenario	Mechanic selects history tab. System fetches past data. Services are listed chronologically.
Alternative Scenario	No completed services available. Data fetch fails. History list shows incorrect entries.

Client Use Case



Login Description

Field	Detail
Goal	Authenticate the client to access personalized services
Initiator	Client
Preconditions	Client has previously registered an account. Client has network connectivity. Client has valid credentials.
Postconditons	Client is logged into the system. A session token is created and stored. Login timestamp is recorded.
Main Success Scenario	Client navigates to the login page. Client enters email and password. System validates credentials against the database System issues session token and redirects to dashboard
Alternative Scenario	Invalid credentials: system displays error message and prompts retry. Network failure: system shows connectivity error. Account locked: system informs client and suggests password reset

Logout Description

Field	Detail
Goal	Terminate the client’s session securely.
Initiator	Client
Preconditions	Client is logged into the system Session is currently active.
Postconditons	Session is terminated. Any sensitive data is cleared from the local cache. Client is returned to the login screen.
Main Success Scenario	Client clicks the "Logout" button. The system ends the session. User is redirected to the login page.
Alternative Scenario	Session timeout occurs before logout is triggered.

	Logout fails due to server timeout. Client mistakenly logs out during an active task.

Search for Mechanic Locations Description

Field	**Detail**
Goal	Find mechanics available in a specified location.
Initiator	Client
Preconditions	Client is logged in. Client has internet access. Location service is operational
Postconditons	System displays a list of mechanics in the area. Mechanic data (name, rating, distance) is retrieved. Search parameters are logged.
Main Success Scenario	Client enters a location name into the search field. Client submits the search request. System queries the mechanic registry for the location. System returns and displays matching mechanic profiles.
Alternative Scenario	No mechanics found: system displays a 'no results' Invalid location input: system prompts for valid location. Service outage: system alerts client to try again later.

Make Order Description

Field	**Detail**
Goal	Place a service order for vehicle assistance.
Initiator	Client
Preconditions	Client is authenticated. Client has selected a mechanic.
Postconditons	Order record is created in the database. Mechanic is notified of new order. Client receives order confirmation.
Main Success Scenario	Client selects a mechanic and service type. Client provides vehicle details and location. System saves the order and notifies mechanic.

	System sends confirmation to client.
Alternative Scenario	Mechanic unavailable: system suggests alternative mechanics. Incomplete details: system prompts for missing information.

View Past Orders Description

Field	**Detail**
Goal	Review previous service orders and statuses
Initiator	Client
Preconditions	Client is logged in. Client has past orders in the system.
Postconditons	List of past orders is displayed. Client can select and view order details. History retrieval is logged.
Main Success Scenario	Client navigates to 'Past Orders' section. System retrieves client's order history. System displays orders with date, mechanic, and status. Client selects an order to view full details.
Alternative Scenario	No past orders: system displays empty state with guidance. Database error: system shows an error page and offers retry.

Give Feedback Description

Field	**Detail**
Goal	Submit feedback on completed service
Initiator	Client
Preconditions	Order status is 'completed'. Client is logged in.
Postconditons	Feedback is stored in the database. Mechanic's rating is updated. Client sees a thank-you confirmation.
Main Success Scenario	Client opens the completed order detail. Client enters rating and comments.

	Client submits feedback. System saves feedback and updates mechanic rating. System displays confirmation message.
Alternative Scenario	Empty feedback: system disables submit button until filled. Server error: system shows error and allows retry later.

UpdatePersonal Information Description

Field	**Detail**
Goal	Allow the client to modify their profile details.
Initiator	Client
Preconditions	Client is authenticated. Client profile exists.
Postconditons	Client's profile data is updated. Change timestamp is recorded. Client sees updated information
Main Success Scenario	Client navigates to profile settings. Client updates fields (name, phone, password). Client submits changes. System validates and saves updates. System confirms success..
Alternative Scenario	Validation error: system highlights invalid fields. Server error: system shows message and retries option..

Delete/Cancel Order Description

Field	**Detail**
Goal	Remove or cancel a pending order.
Initiator	Client
Preconditions	Order status is 'pending'. Client is authenticated.
Postconditons	Order status is updated to 'cancelled'. Mechanic is notified of cancellation. Client receives cancellation confirmation

Main Success Scenario	Client views their pending orders. Client selects an order and chooses 'Cancel'. System updates order status to 'cancelled'. System notifies mechanic and client.
Alternative Scenario	Order already in service: system prevents cancellation. Notification failure: system logs error and alerts client.

Share Current Location Description

Field	**Detail**
Goal	Provide real-time GPS location to the system
Initiator	Client
Preconditions	Client’s device GPS is enabled. Client grants location permission.
Postconditons	Current coordinates are saved. Mechanic sees updated location in order detail.
Main Success Scenario	Client opens sharing modal. Client taps 'Share Location'. System retrieves GPS coordinates. System updates order location.
Alternative Scenario	Permission denied: system prompts permission request. GPS unavailable: system displays error message.

Search by Nearest Mechanic Description

Field	**Detail**
Goal	Automatically find the closest mechanic.
Initiator	Client
Preconditions	Client has shared location. Mechanic locations database is up-to-date.
Postconditons	Nearest mechanic list is displayed. Client can select one for order.
Main Success Scenario	Client taps 'Find Nearest'. System calculates distances to mechanics.

	System sorts by proximity and returns list.
Alternative scenario	No mechanics within radius: system expands search radius. Calculation error: system shows fallback list.

View Mechanic Details &Get Directions Description

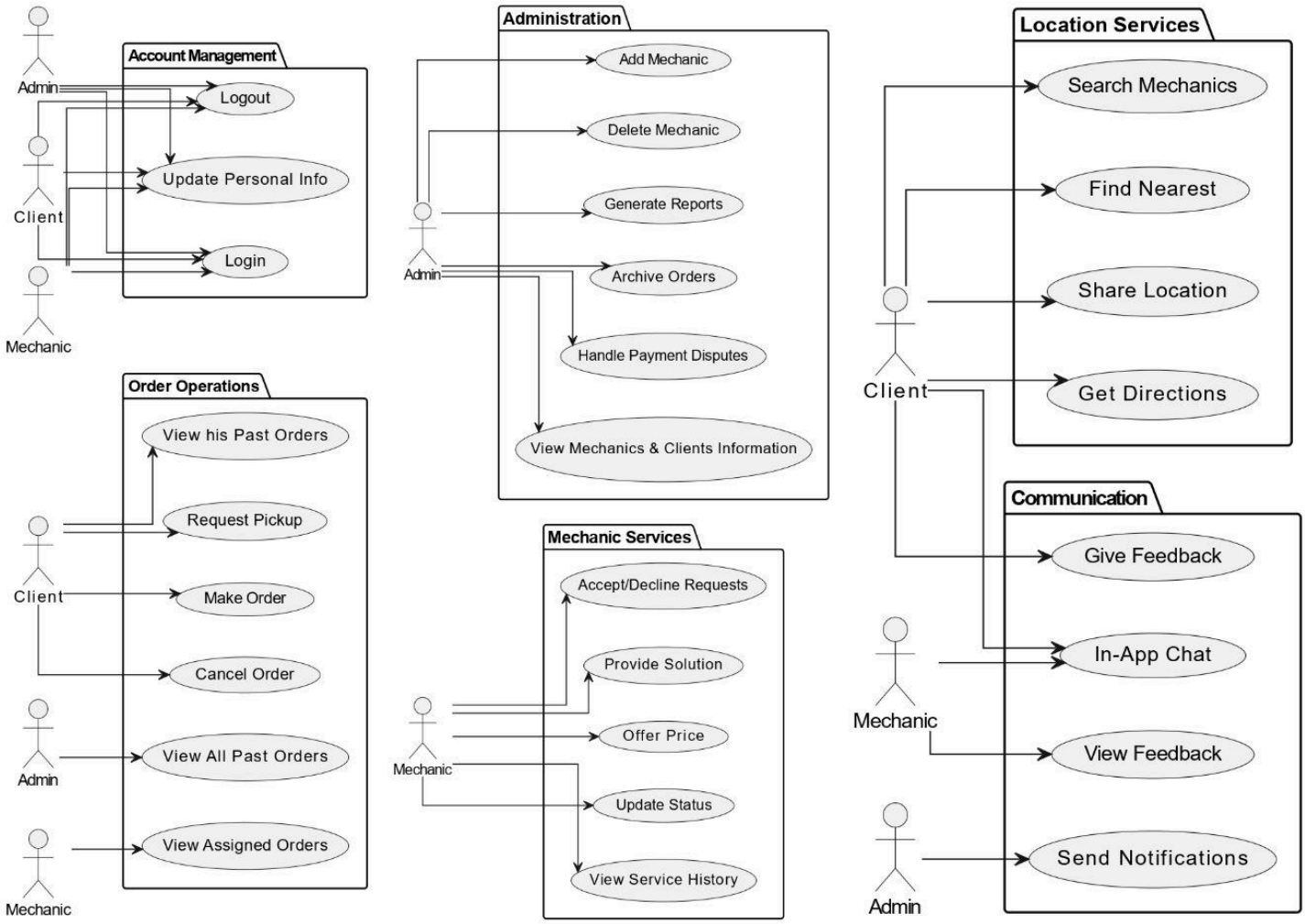
Field	**Detail**
Goal	See a mechanic's profile and navigate to their location.
Initiator	Client
Preconditions	Client has searched or selected a mechanic. Map service is available.
Postconditons	Mechanic’s profile displays correctly. Directions link is available.
Main Success Scenario	Client selects a mechanic from list. System shows mechanic details (ratings, services). Client clicks 'Get Directions'. System opens map with route
Alternative Scenario	Map service error: system shows text-based directions. Profile data missing: system shows placeholder info.

Request On-Spot Pickup Description

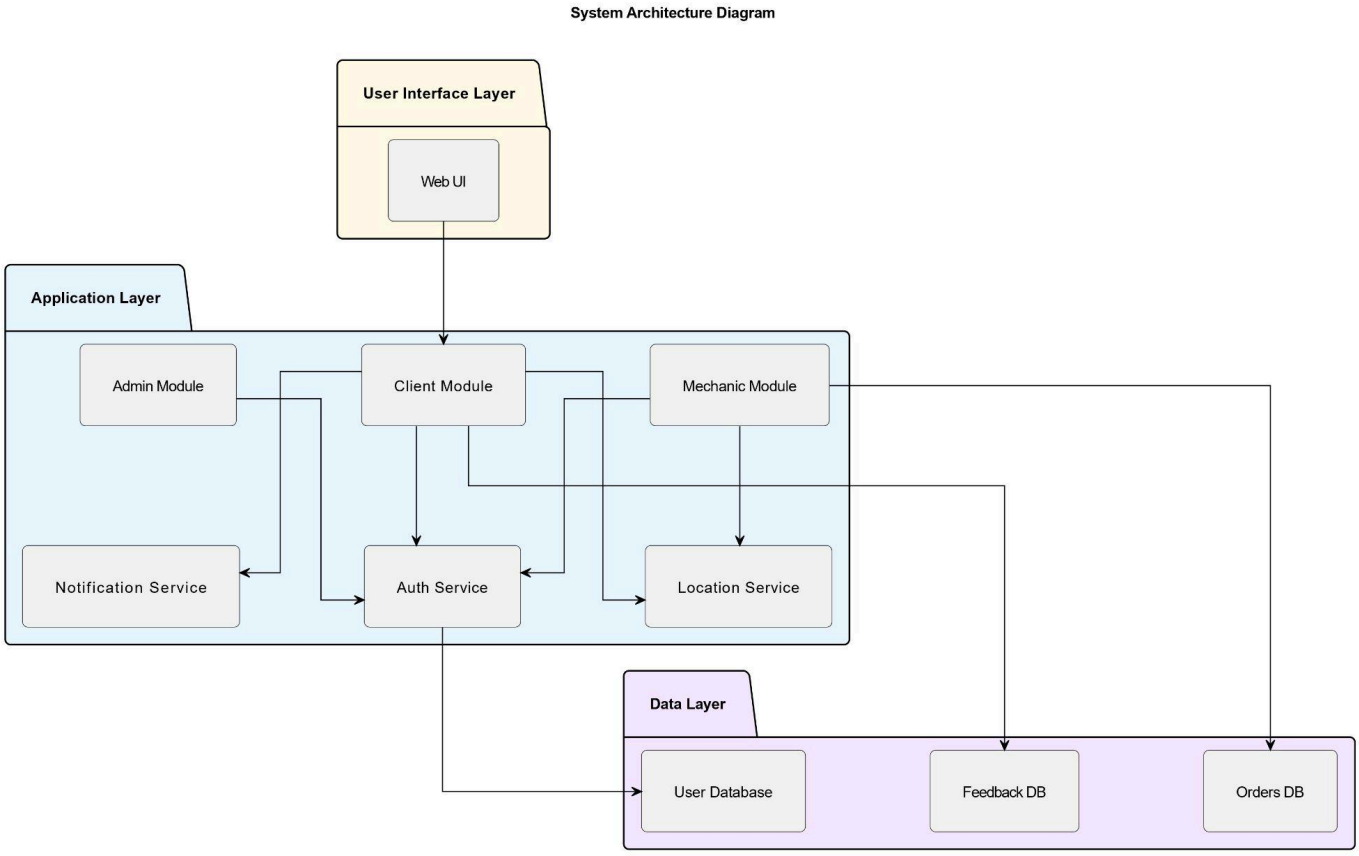
Field	**Detail**
Goal	Request a mechanic to come to the client's vehicle
Initiator	Client
Preconditions	Client has made an order. Mechanic is assigned. Client location is known.
Postconditons	Mechanic receives pickup request. Client sees pickup ETA.
Main Success Scenario	Client clicks 'Request Pickup'. System sends pickup request to mechanic. Mechanic acknowledges pickup. System displays ETA to client.

Alternative Scenario	Mechanic does not respond: system suggests reassigning order. Network error: system prompts retry.

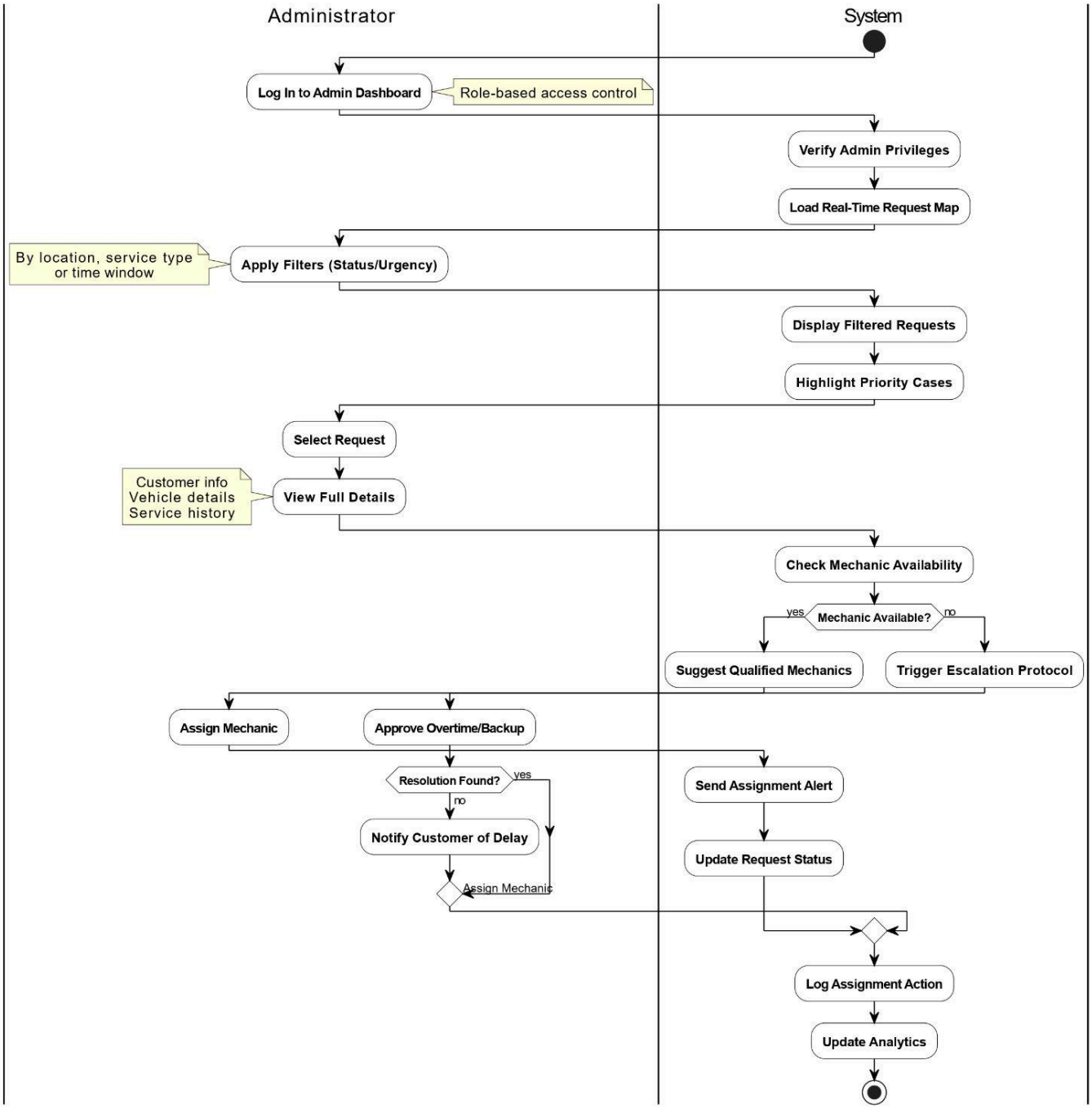
Use Case Package Diagram:



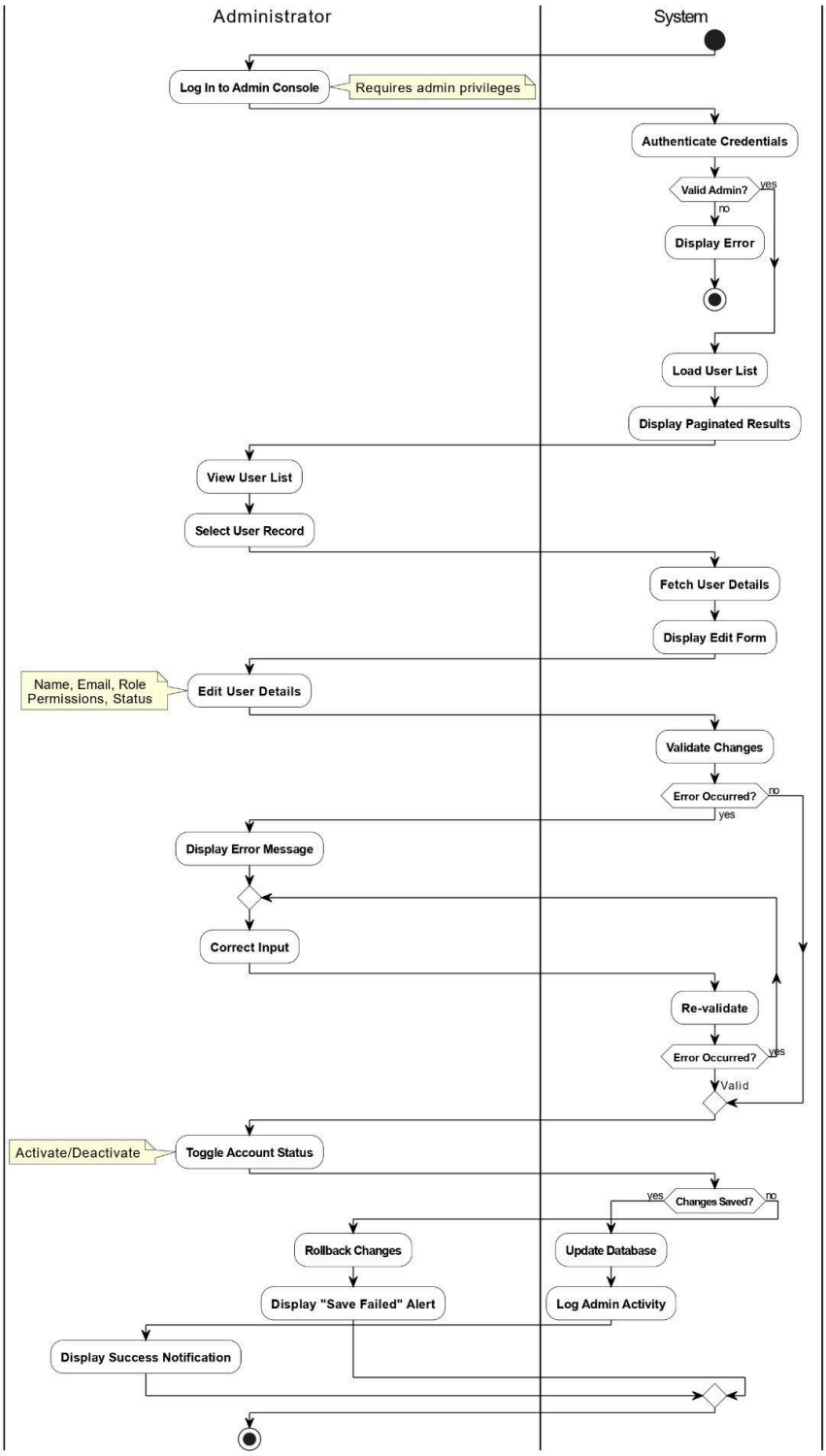
System Architecture:



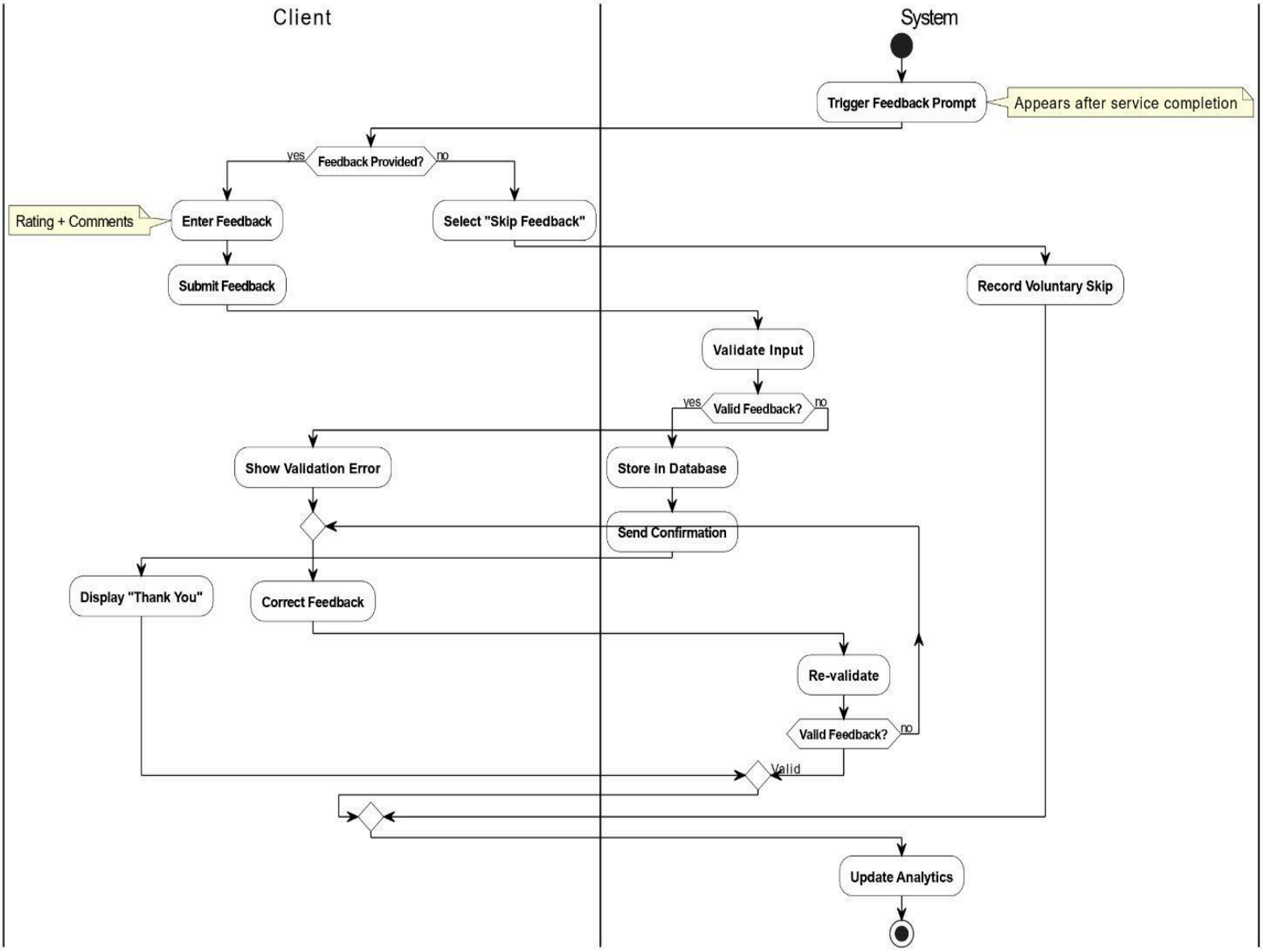
Activity Diagrams
Admin Monitoring Service Requests



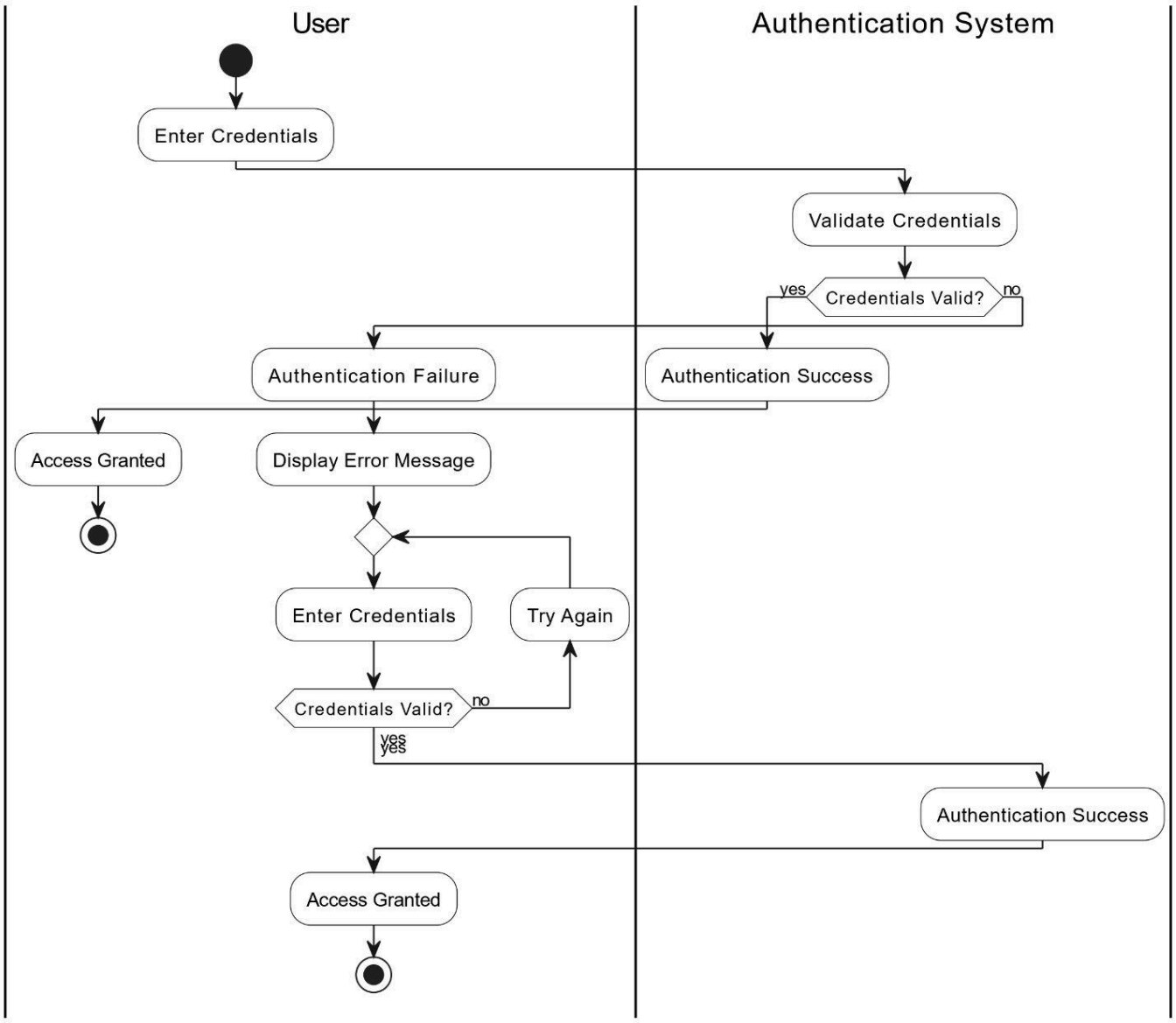
Admin User Management



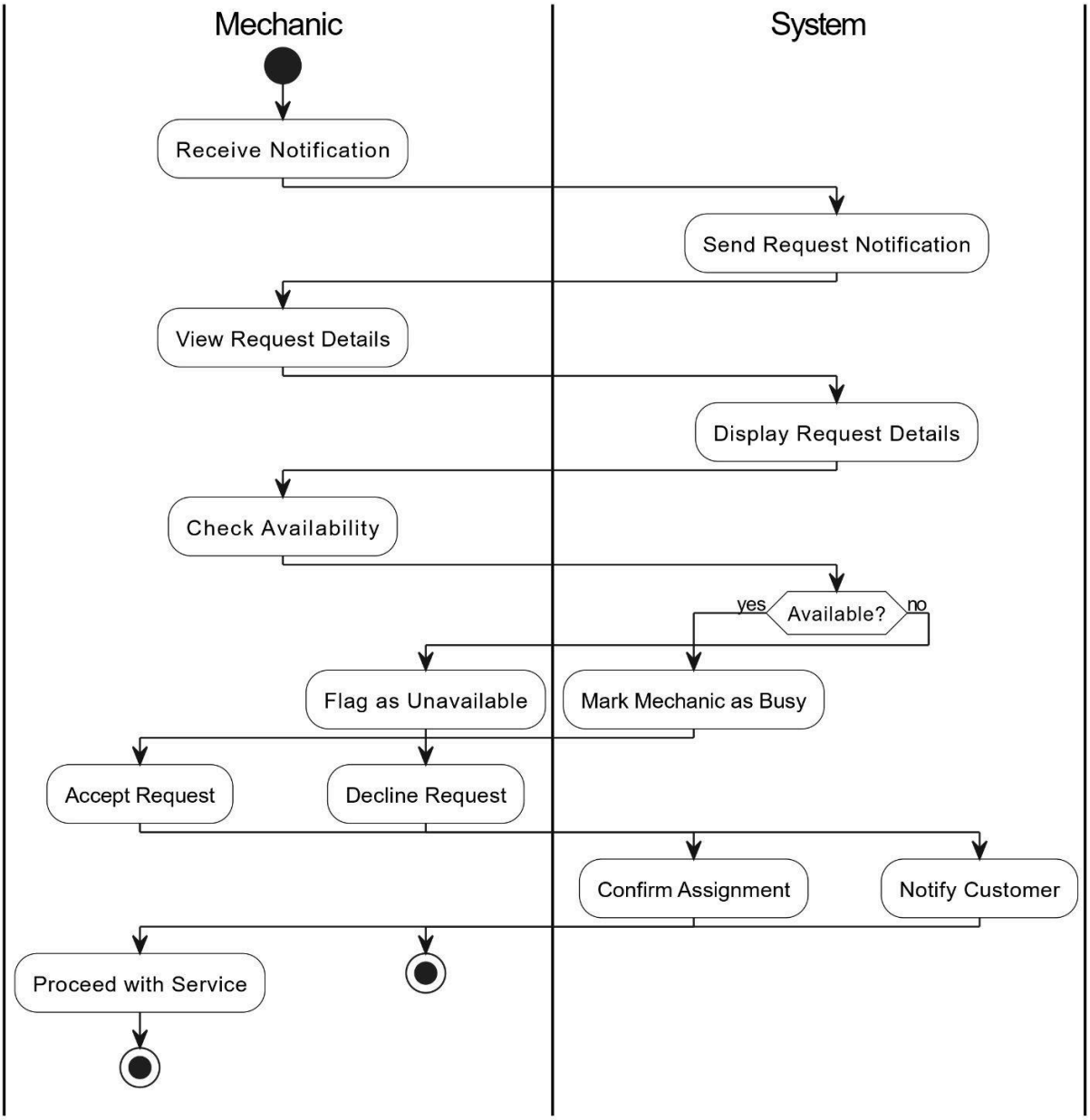
Feedback Submission



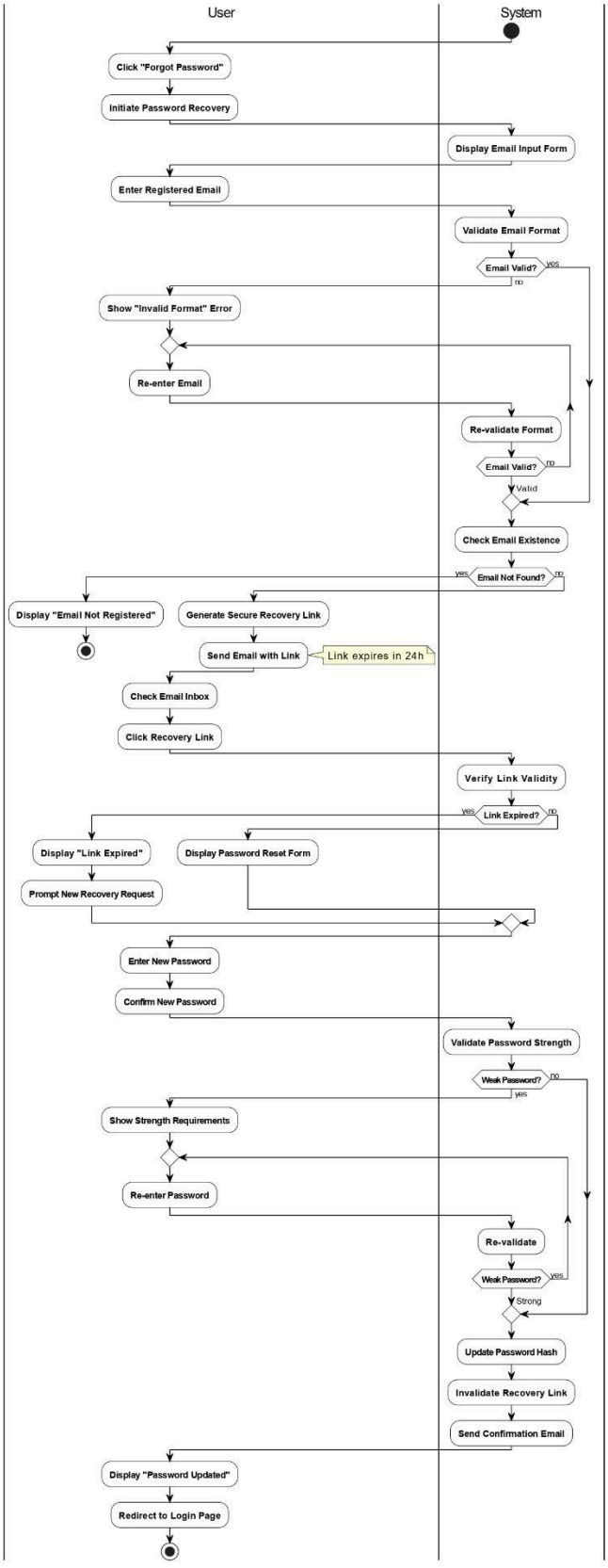
Login and Authentication



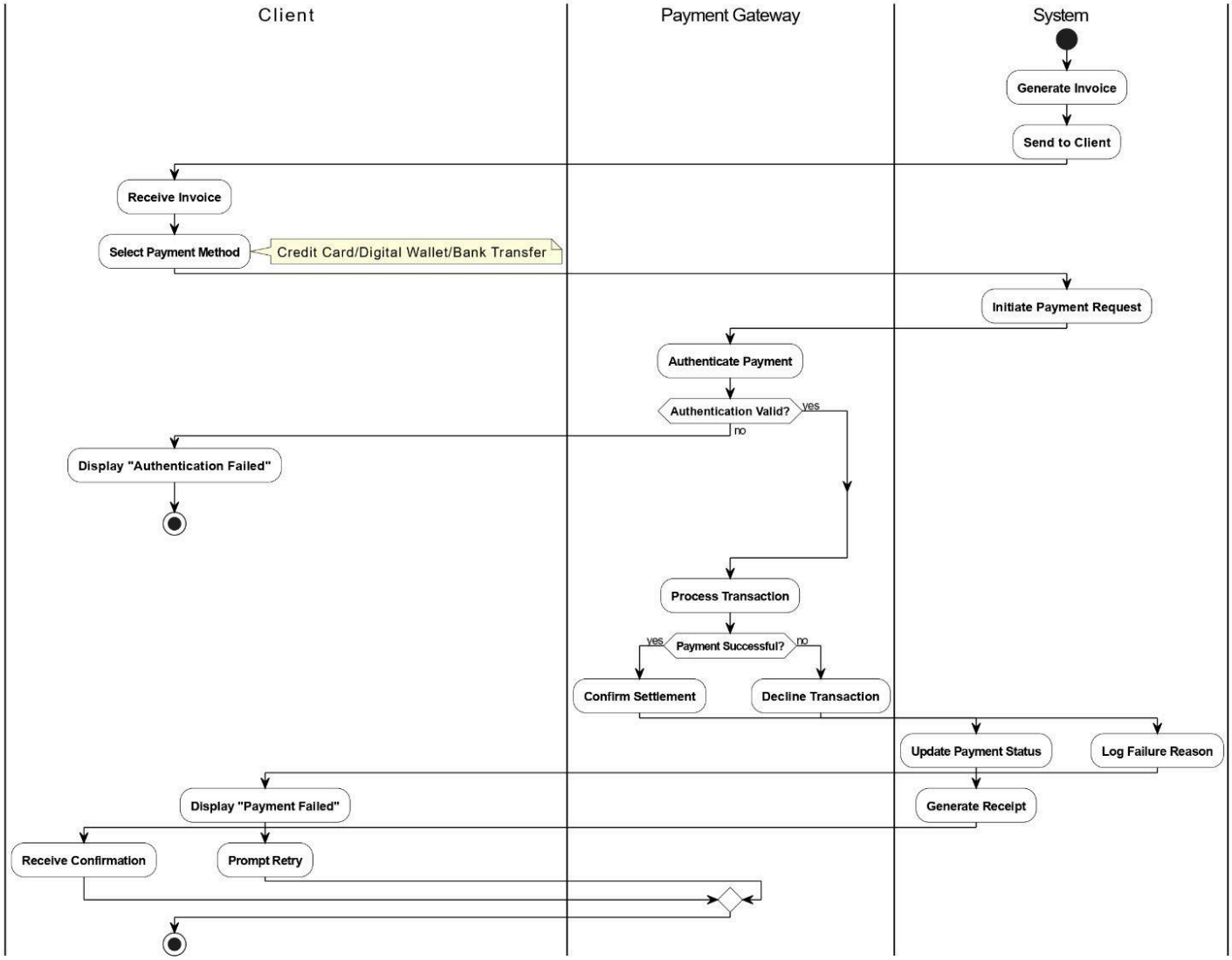
Mechanic Accepting a Service Request



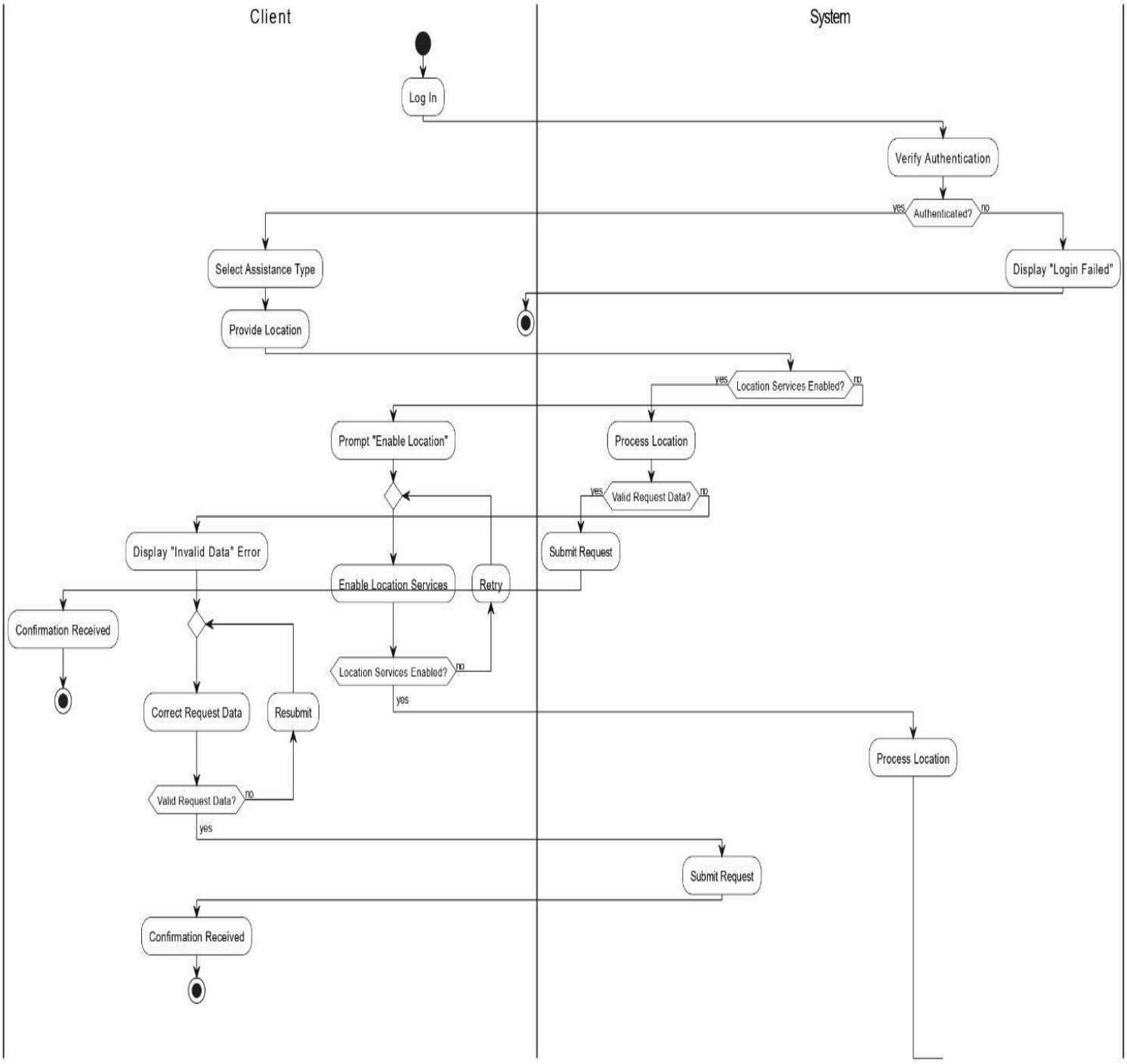
Password Recovery Process



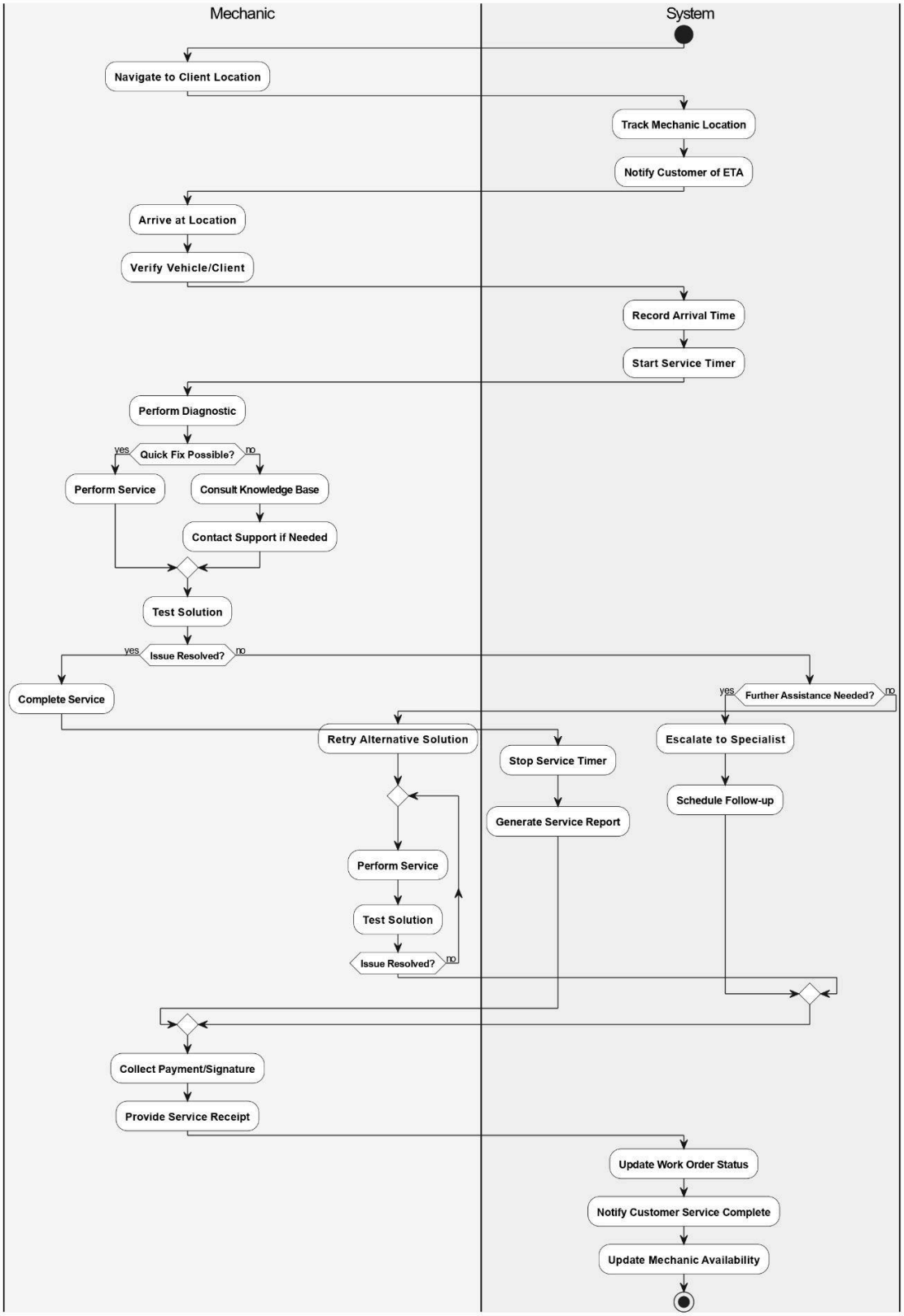
Payment Processing



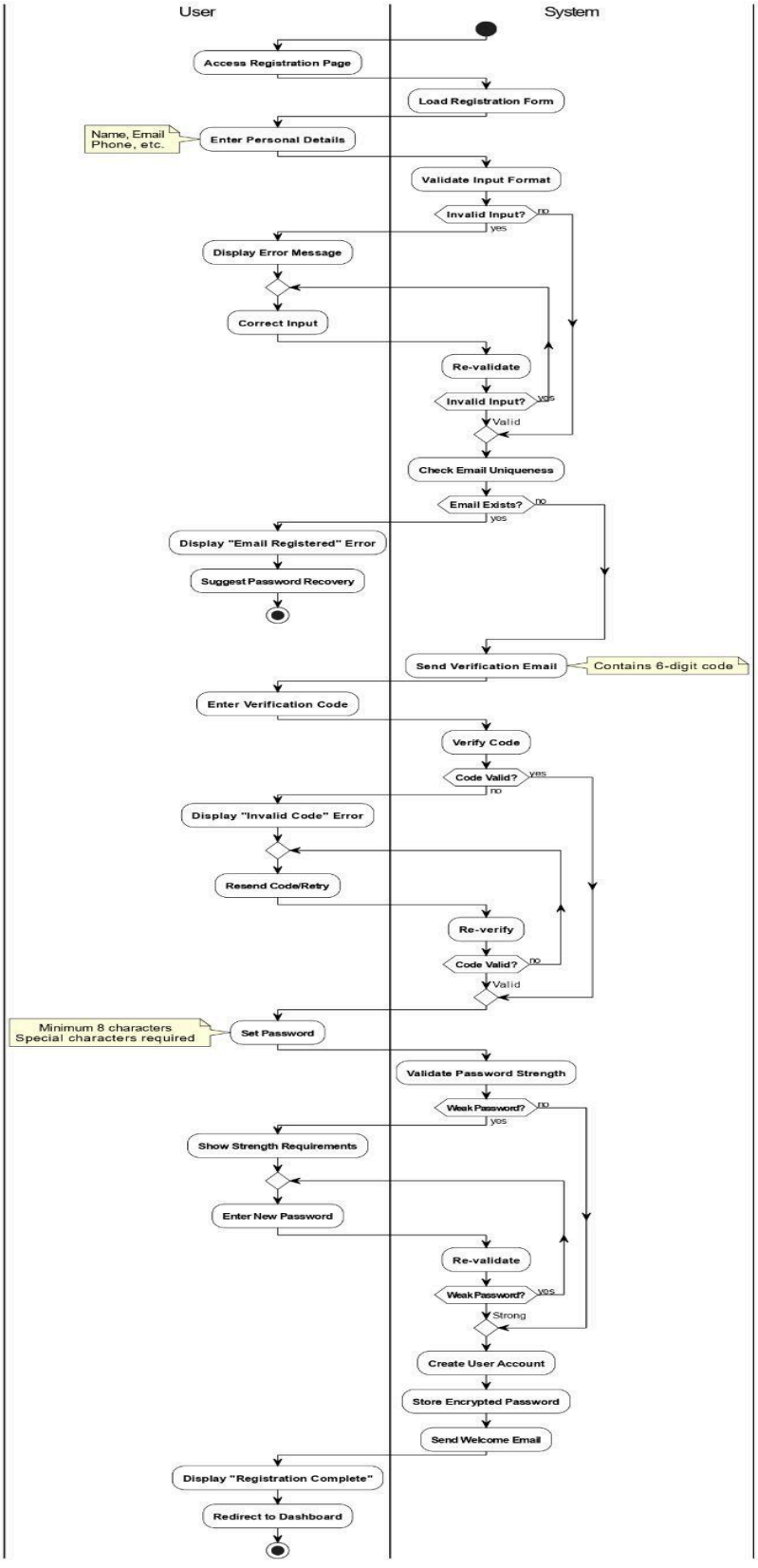
Requesting Roadside Assistance



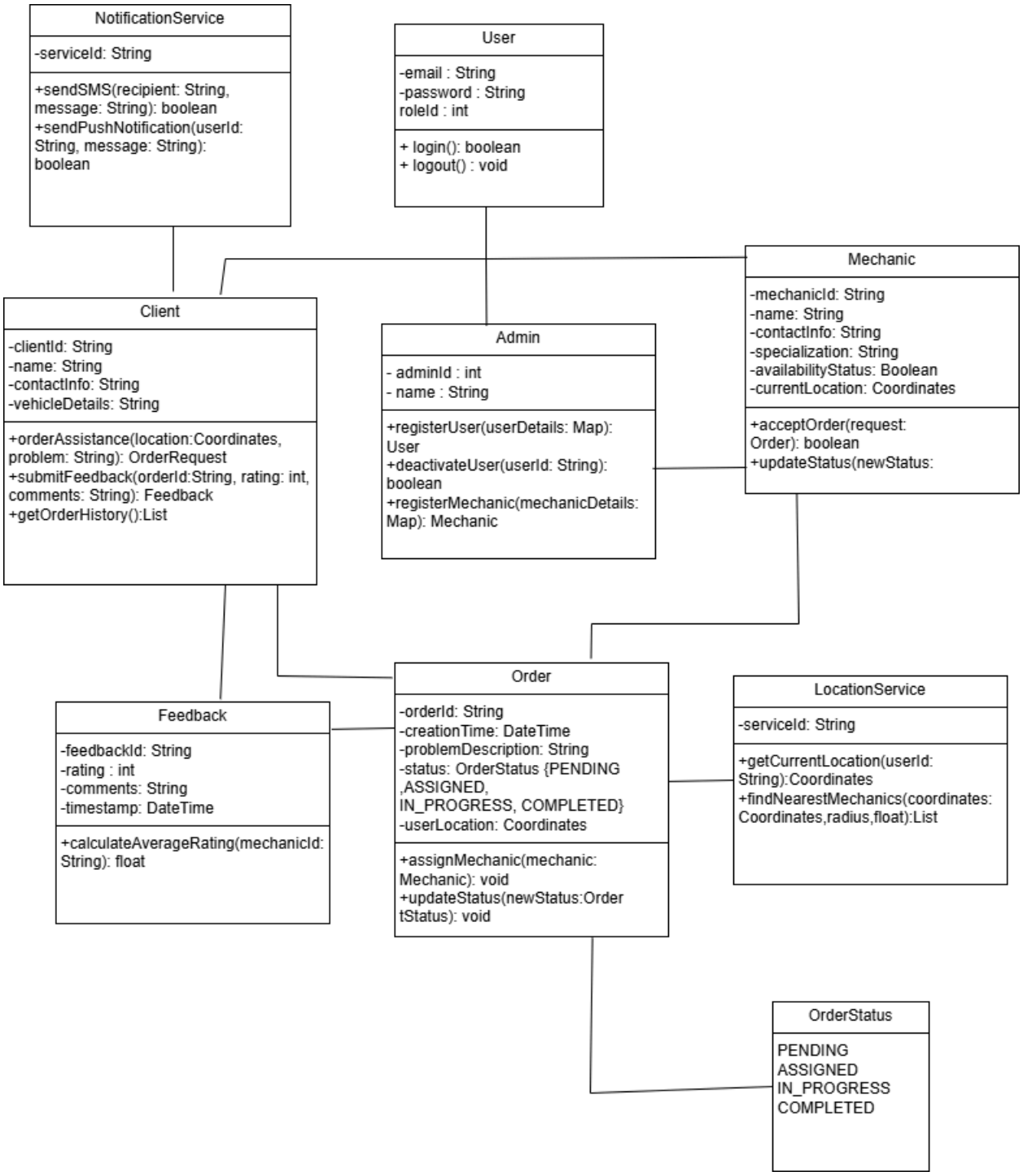
Service Fulfillment Process



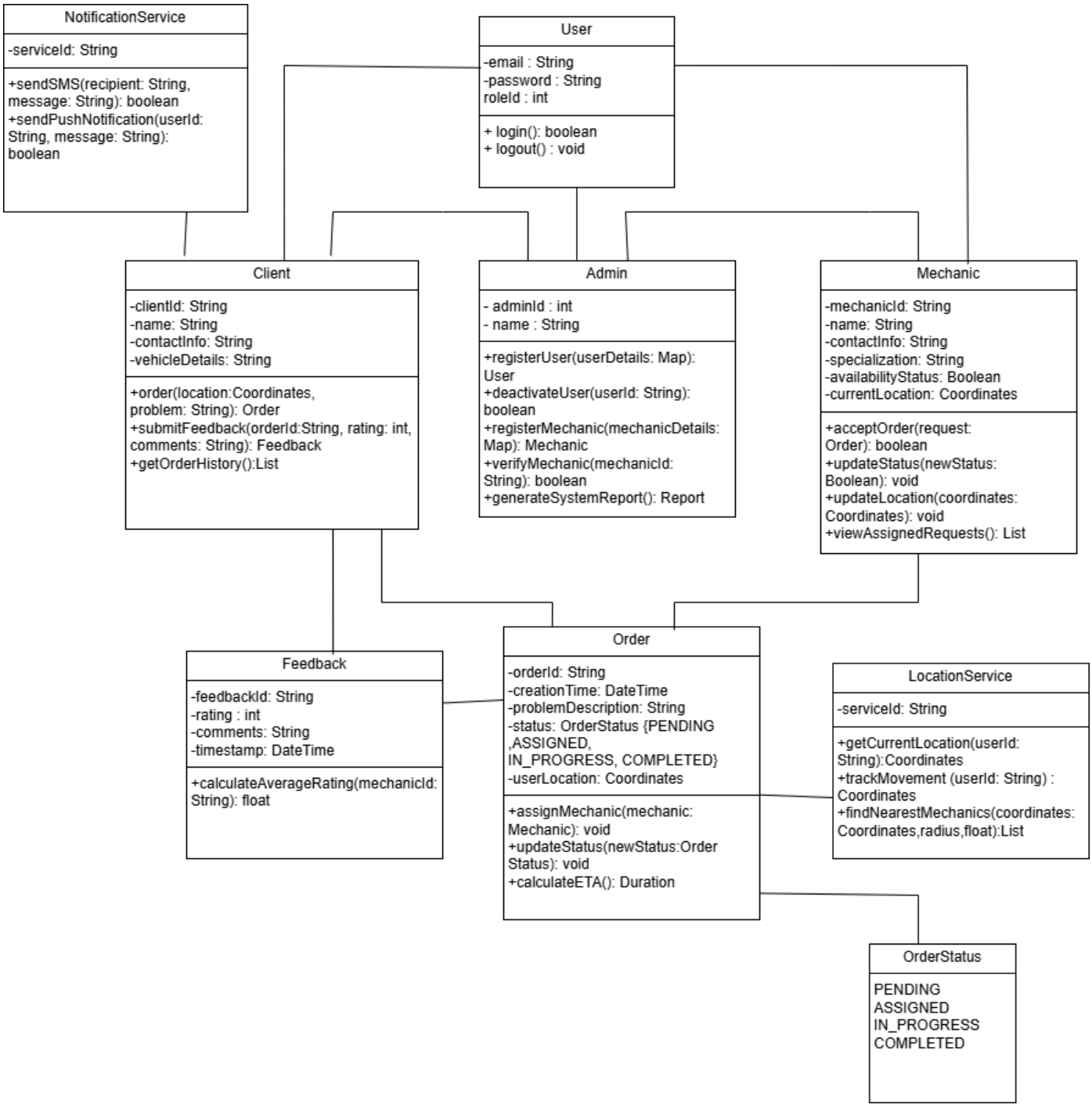
User Registration Process



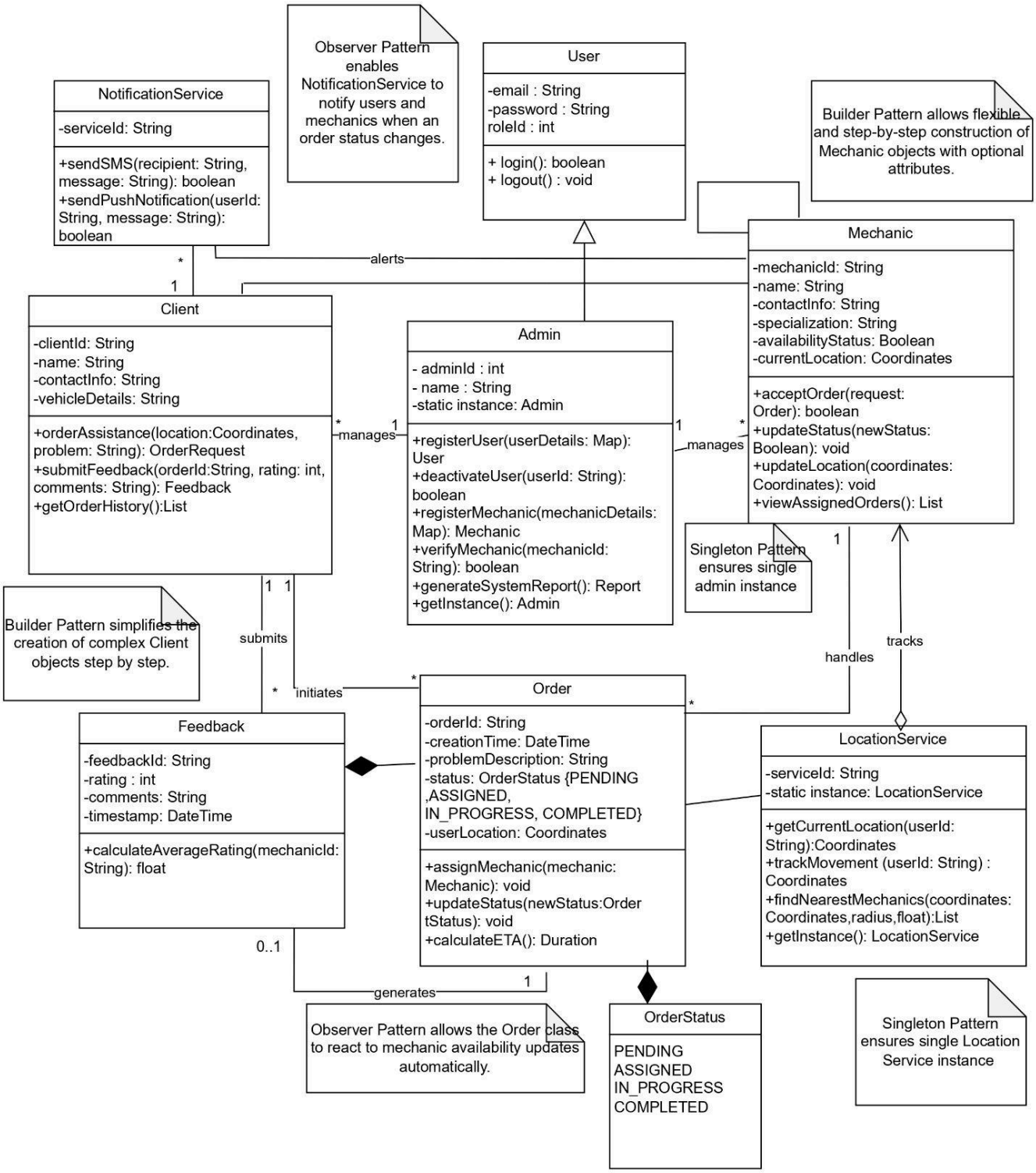
Class Diagrams
Version 1



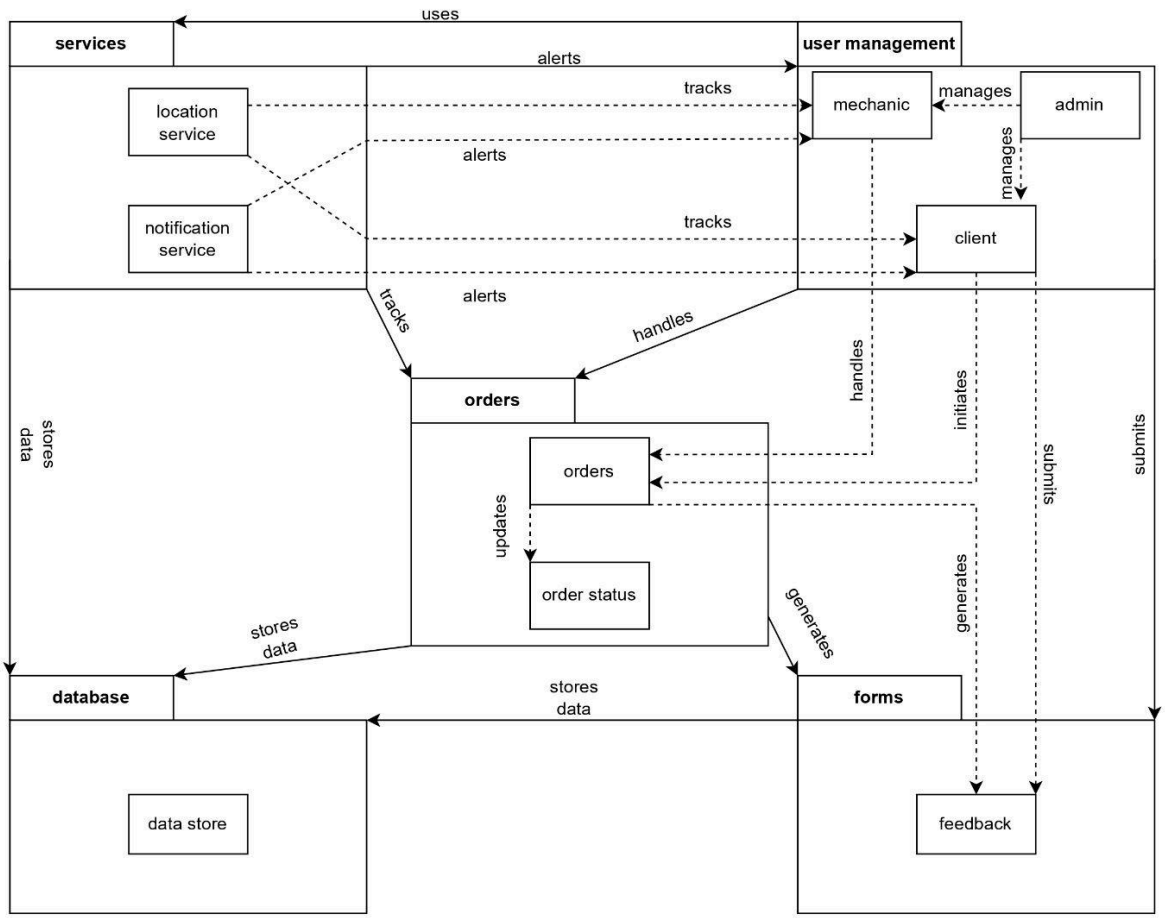
Version 2


NotificationService
-serviceId: String
+sendSMS(recipient: String, message: String): boolean
+sendPushNotification(userId: String, message: String): boolean
User
-email : String
-password : String
roleId : int
+ login(): boolean
+ logout() : void
Client
-clientId: String
-name: String
-contactInfo: String
-vehicleDetails: String
+order(location:Coordinates, problem: String): Order
+submitFeedback(orderId:String, rating: int, comments: String): Feedback
+getOrderHistory():List
Admin
- adminId : int
- name : String
+registerUser(userDetails: Map): User
+deactivateUser(userId: String): boolean
+registerMechanic(mechanicDetails: Map): Mechanic
+verifyMechanic(mechanicId: String): boolean
+generateSystemReport(): Report
Mechanic
-mechanicId: String
-name: String
-contactInfo: String
-specialization: String
-availabilityStatus: Boolean
-currentLocation: Coordinates
+acceptOrder(request: Order): boolean
+updateStatus(newStatus: Boolean): void
+updateLocation(coordinates: Coordinates): void
+viewAssignedRequests(): List
Feedback
-feedbackId: String
-rating : int
-comments: String
-timestamp: DateTime
+calculateAverageRating(mechanicId: String): float
Order
-orderId: String
-creationTime: DateTime
-problemDescription: String
-status: OrderStatus {PENDING ,ASSIGNED, IN_PROGRESS, COMPLETED}
-userLocation: Coordinates
+assignMechanic(mechanic: Mechanic): void
+updateStatus(newStatus:Order Status): void
+calculateETA(): Duration
LocationService
-serviceId: String
+getCurrentLocation(userId: String):Coordinates
+trackMovement (userId: String) : Coordinates
+findNearestMechanics(coordinates: Coordinates,radius,float):List
OrderStatus
PENDING
ASSIGNED
IN_PROGRESS
COMPLETED

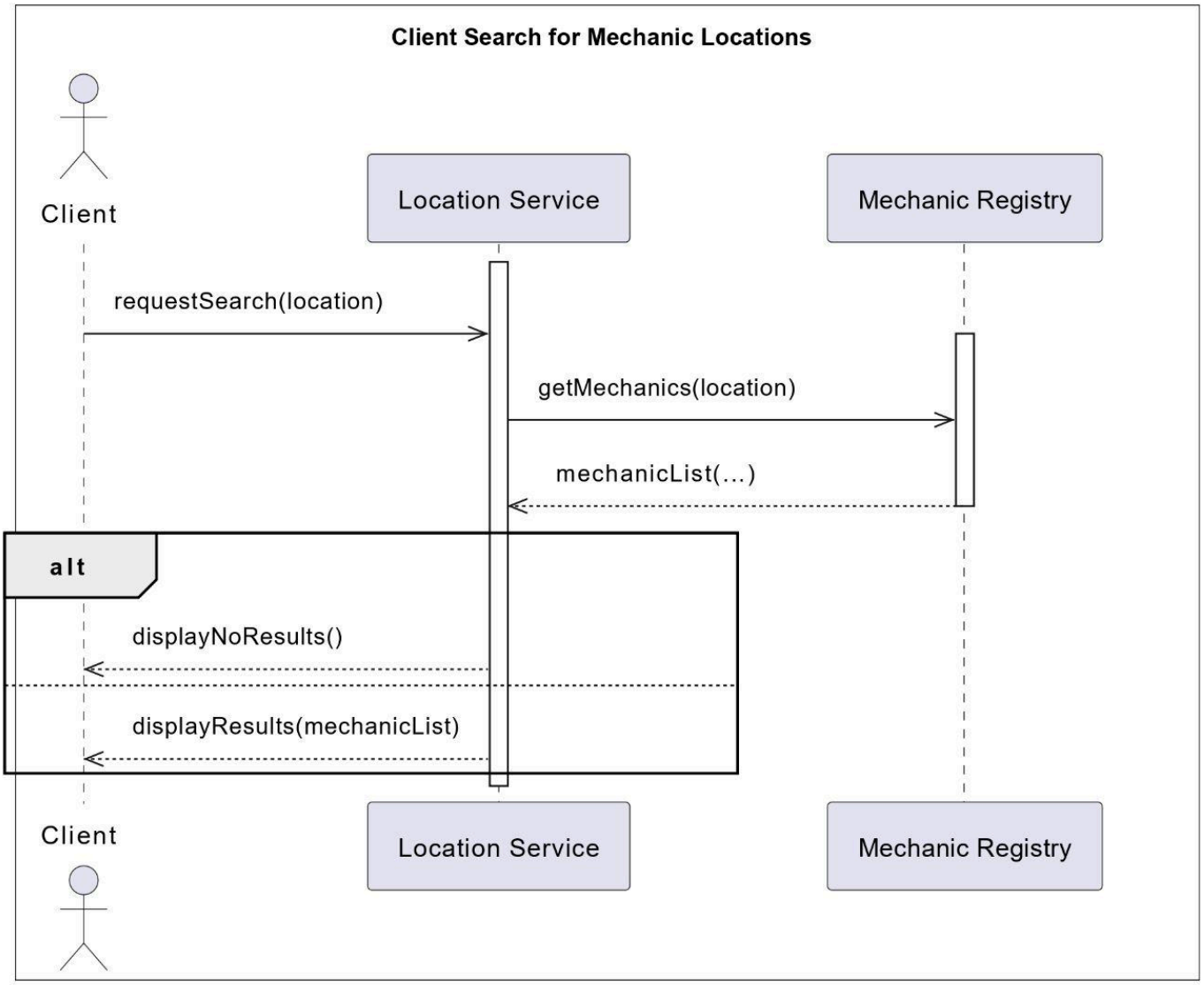
Version 3

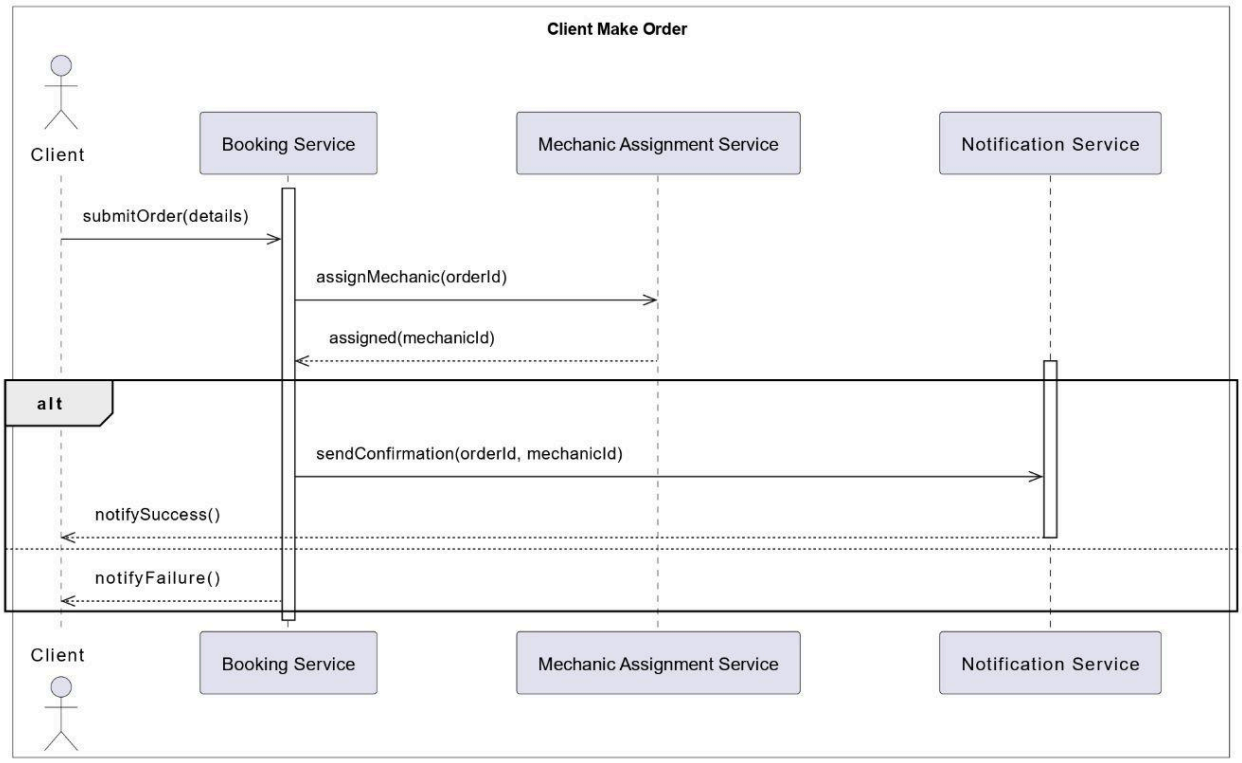

Observer Pattern enables NotificationService to notify users and mechanics when an order status changes.
NotificationService
-serviceId: String
+sendSMS(recipient: String, message: String): boolean
+sendPushNotification(userId: String, message: String): boolean
User
-email : String
-password : String
roleId : int
+ login(): boolean
+ logout() : void
Builder Pattern allows flexible and step-by-step construction of Mechanic objects with optional attributes.
Mechanic
-mechanicId: String
-name: String
-contactInfo: String
-specialization: String
-availabilityStatus: Boolean
-currentLocation: Coordinates
+acceptOrder(request: Order): boolean
+updateStatus(newStatus: Boolean): void
+updateLocation(coordinates: Coordinates): void
+viewAssignedOrders(): List
alerts
Client
-clientId: String
-name: String
-contactInfo: String
-vehicleDetails: String
+orderAssistance(location:Coordinates, problem: String): OrderRequest
+submitFeedback(orderId:String, rating: int, comments: String): Feedback
+getOrderHistory():List
Admin
- adminId : int
- name : String
-static instance: Admin
+registerUser(userDetails: Map): User
+deactivateUser(userId: String): boolean
+registerMechanic(mechanicDetails: Map): Mechanic
+verifyMechanic(mechanicId: String): boolean
+generateSystemReport(): Report
+getInstance(): Admin
manages
manages
Singleton Pattern ensures single admin instance
Builder Pattern simplifies the creation of complex Client objects step by step.
submits
initiates
handles
tracks
Order
-orderId: String
-creationTime: DateTime
-problemDescription: String
-status: OrderStatus {PENDING ,ASSIGNED, IN_PROGRESS, COMPLETED}
-userLocation: Coordinates
+assignMechanic(mechanic: Mechanic): void
+updateStatus(newStatus:OrdertStatus): void
+calculateETA(): Duration
Feedback
-feedbackId: String
-rating : int
-comments: String
-timestamp: DateTime
+calculateAverageRating(mechanicId: String): float
LocationService
-serviceId: String
-static instance: LocationService
+getCurrentLocation(userId: String):Coordinates
+trackMovement (userId: String) : Coordinates
+findNearestMechanics(coordinates: Coordinates,radius,float):List
+getInstance(): LocationService
generates
Observer Pattern allows the Order class to react to mechanic availability updates automatically.
OrderStatus
PENDING
ASSIGNED
IN_PROGRESS
COMPLETED
Singleton Pattern ensures single Location Service instance

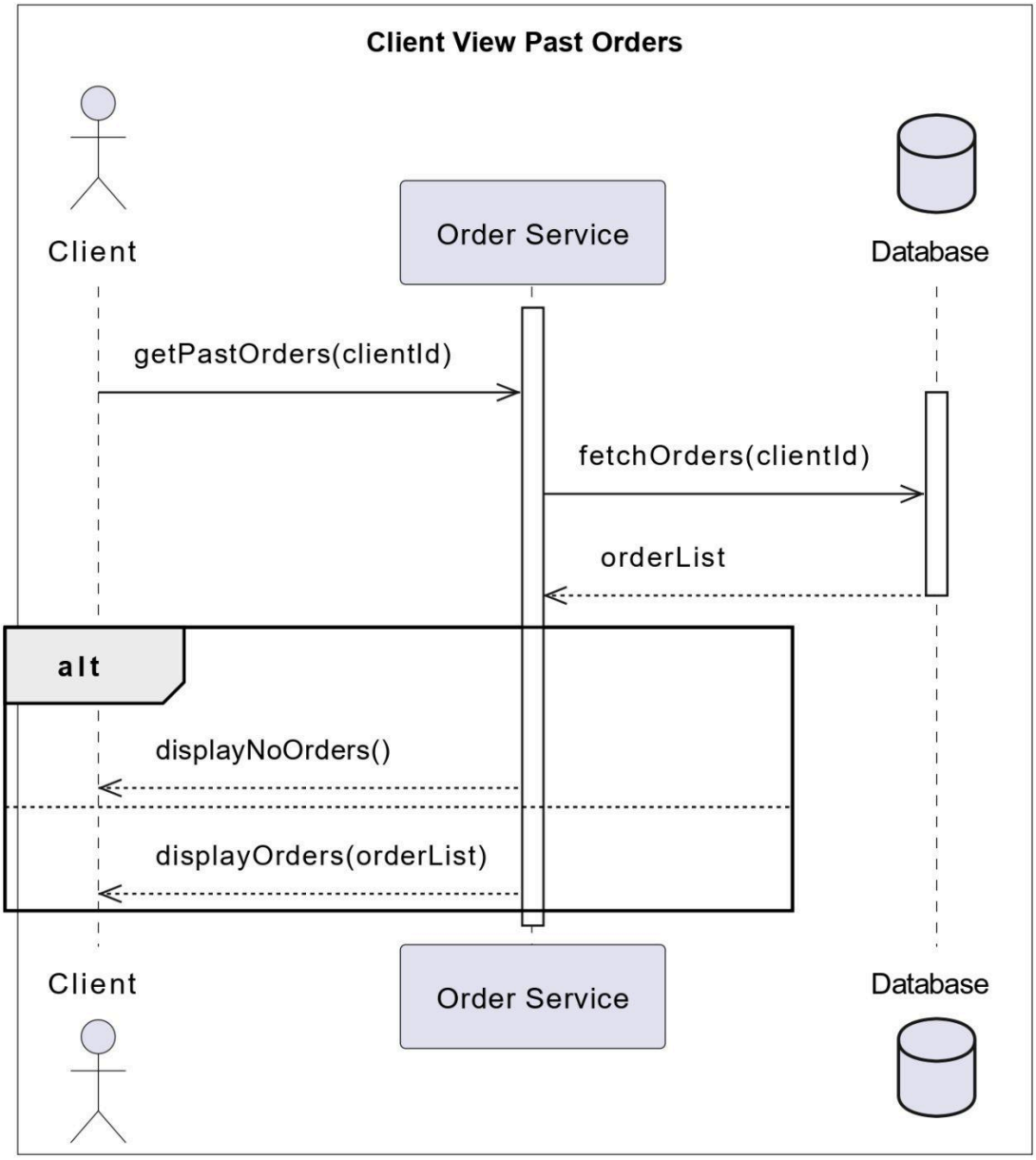
Package Diagram grouping relevant Classes into Packages.

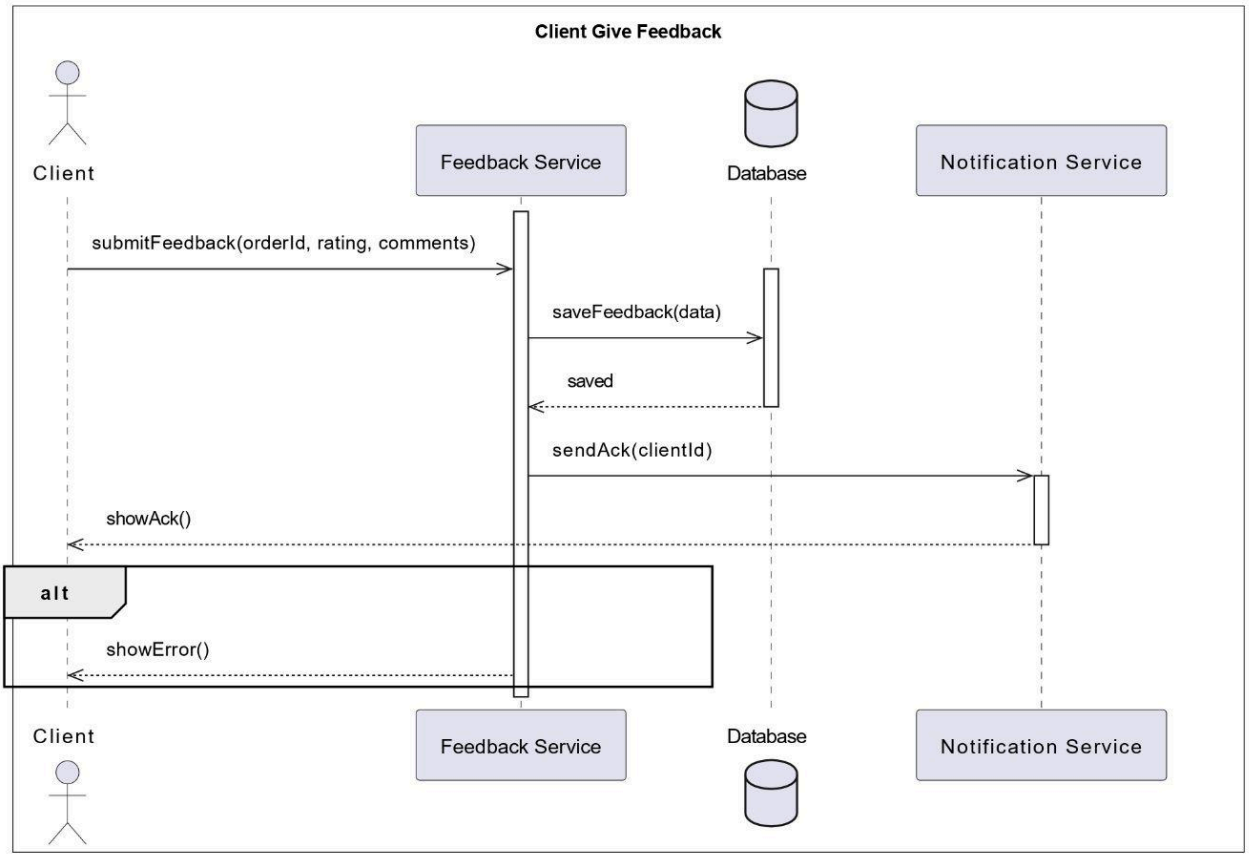


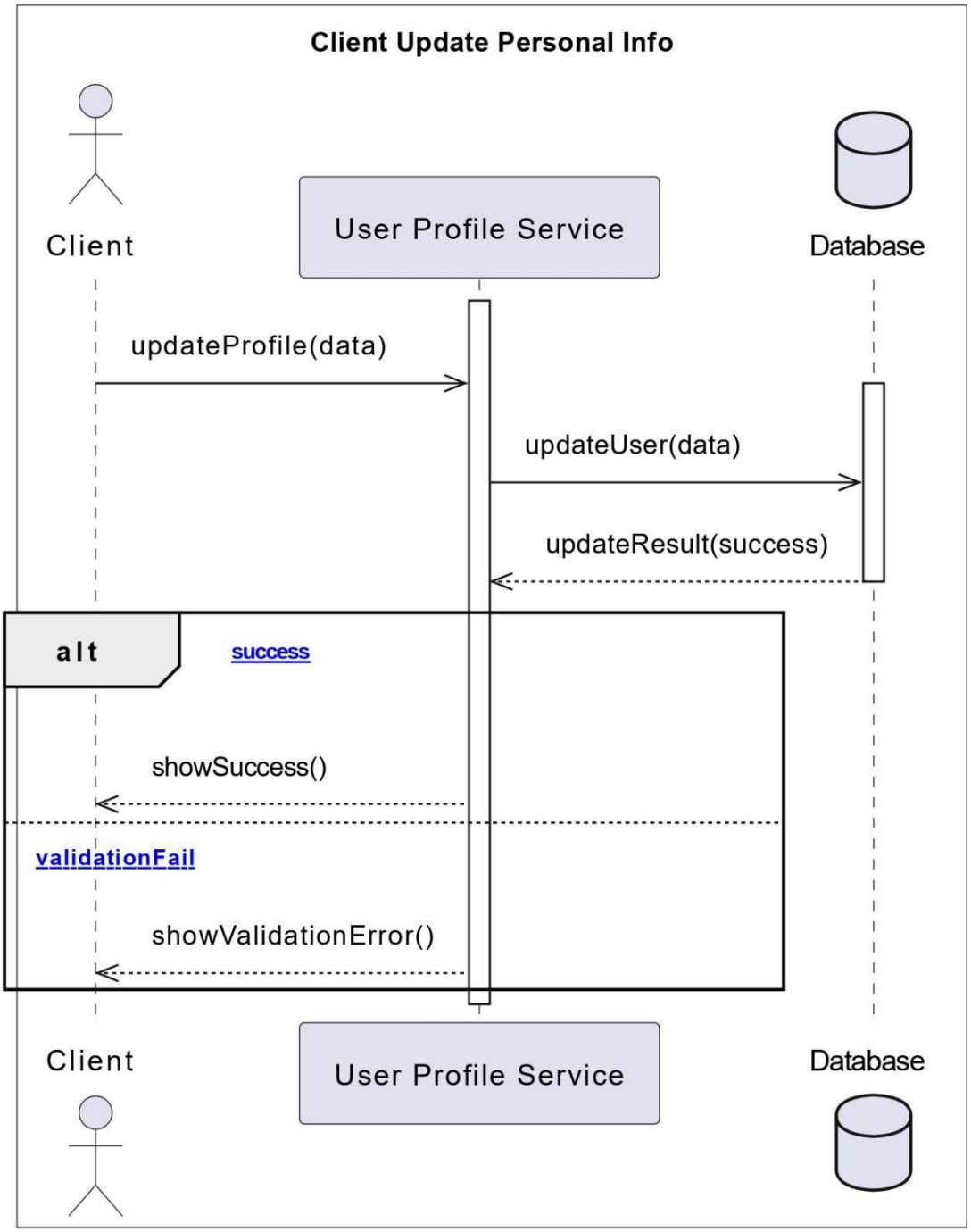
Sequence Diagrams
Client sequence diagrams

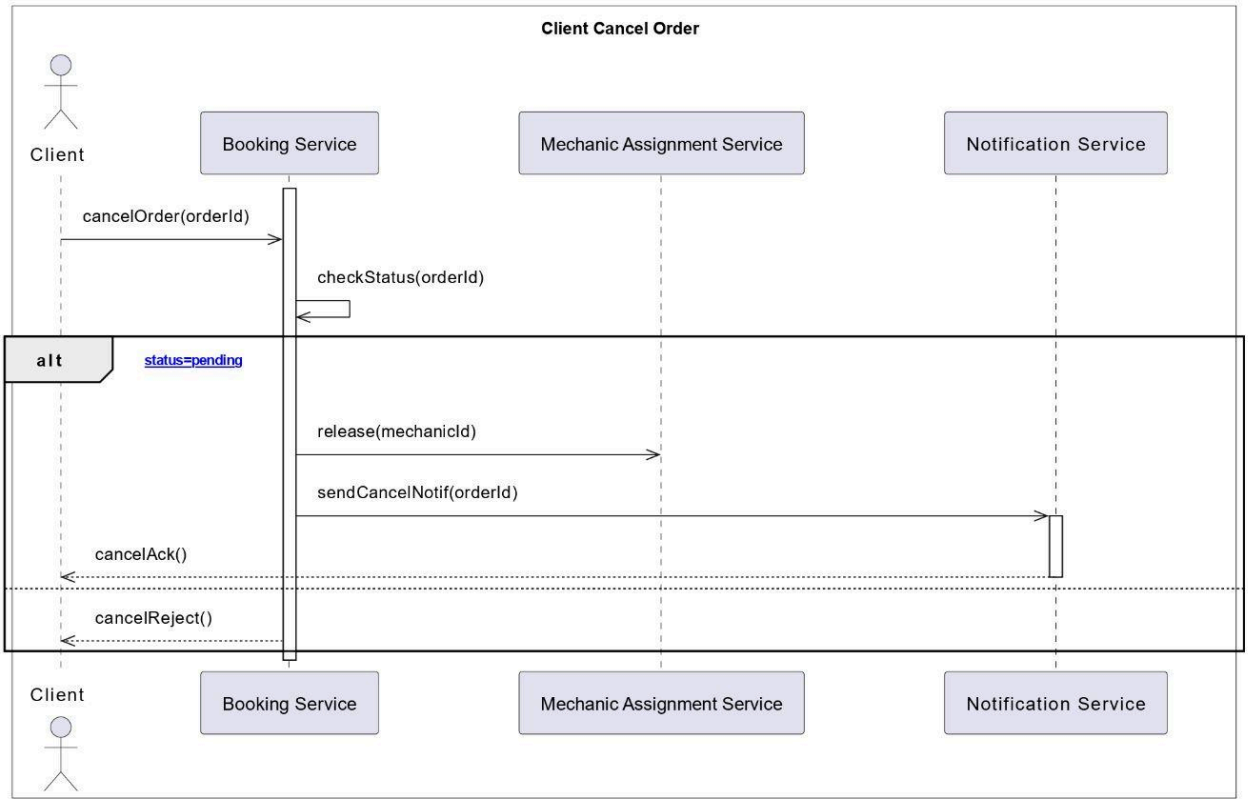


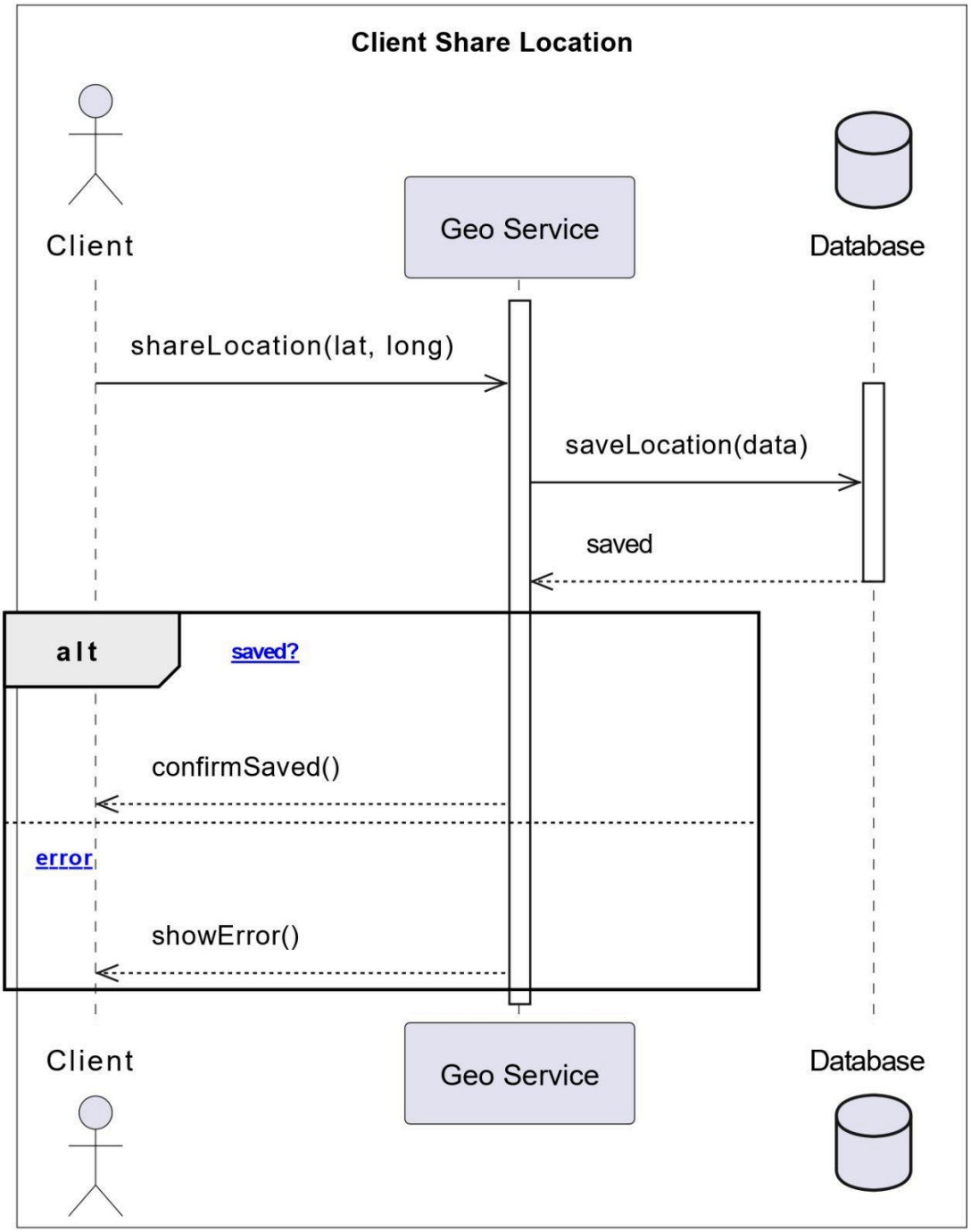

Client Make Order
Client
Booking Service
Mechanic Assignment Service
Notification Service
submitOrder(details)
assignMechanic(orderId)
assigned(mechanicId)
alt
sendConfirmation(orderId, mechanicId)
notifySuccess()
notifyFailure()
Client
Booking Service
Mechanic Assignment Service
Notification Service

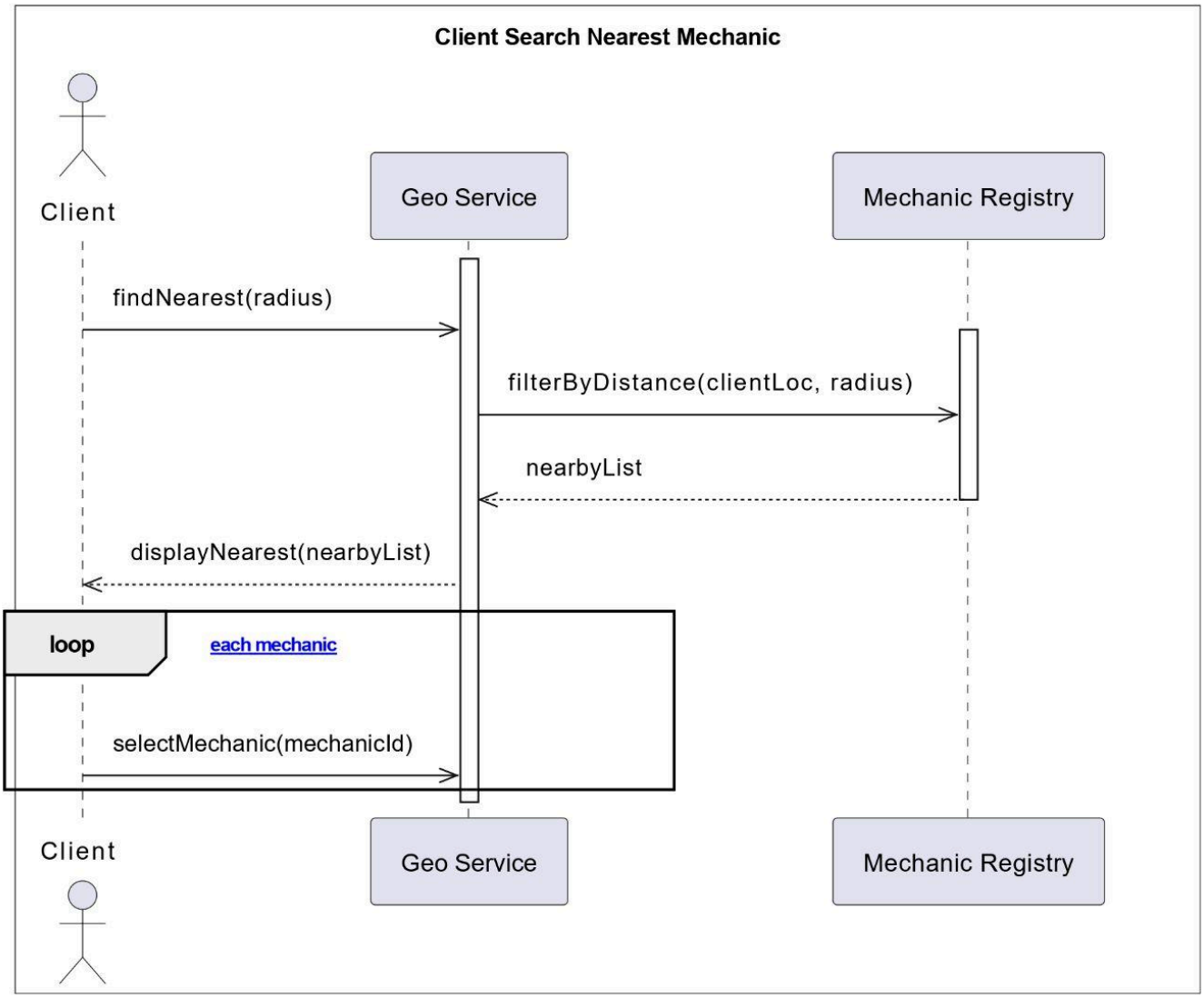

Client View Past Orders
Client
Order Service
Database
getPastOrders(clientId)
fetchOrders(clientId)
orderList
alt
displayNoOrders()
displayOrders(orderList)
Client
Order Service
Database

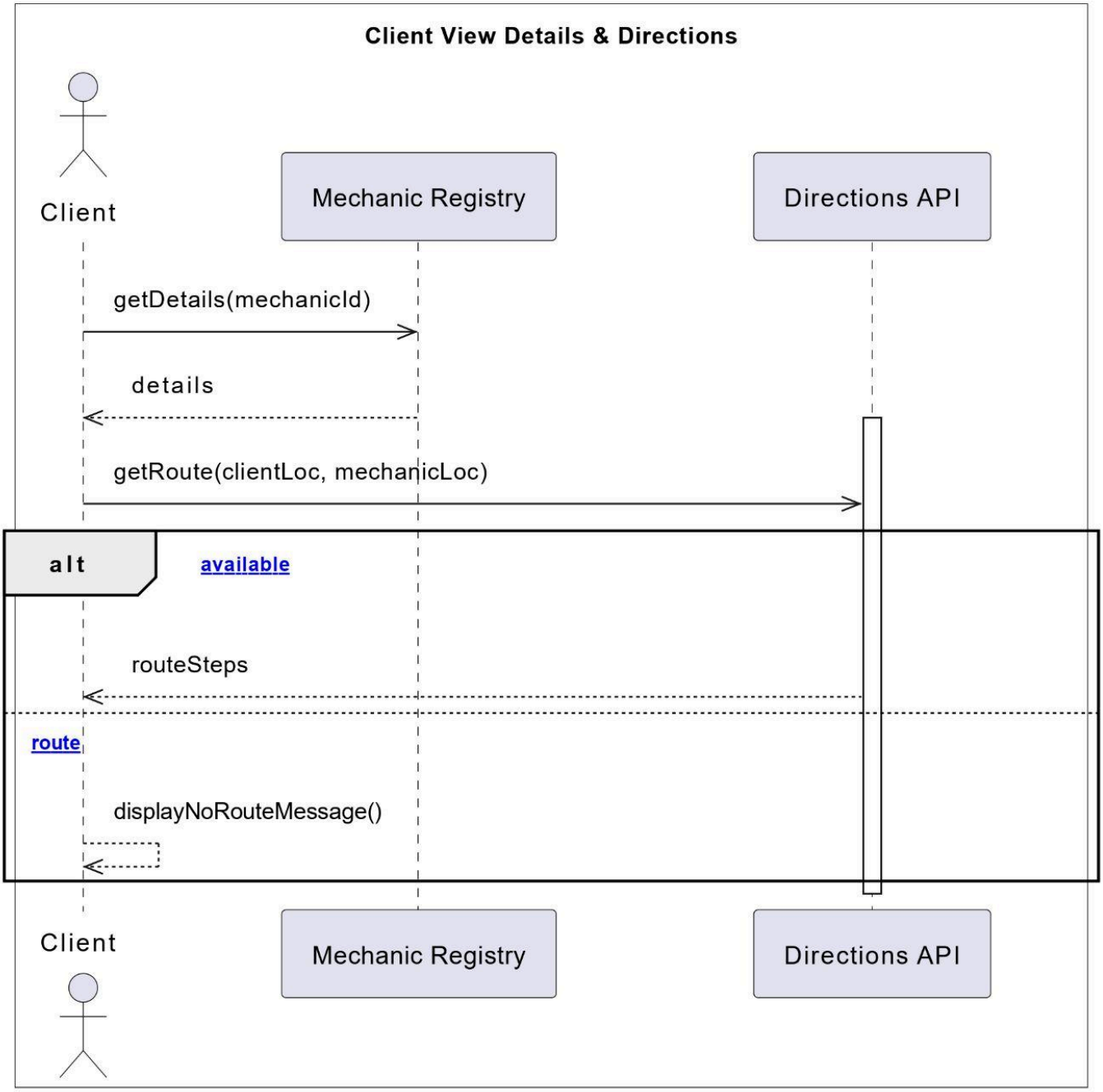

Client Give Feedback
Client
Feedback Service
Database
Notification Service
submitFeedback(orderId, rating, comments)
saveFeedback(data)
saved
sendAck(clientId)
showAck()
alt
showError()
Client
Feedback Service
Database
Notification Service

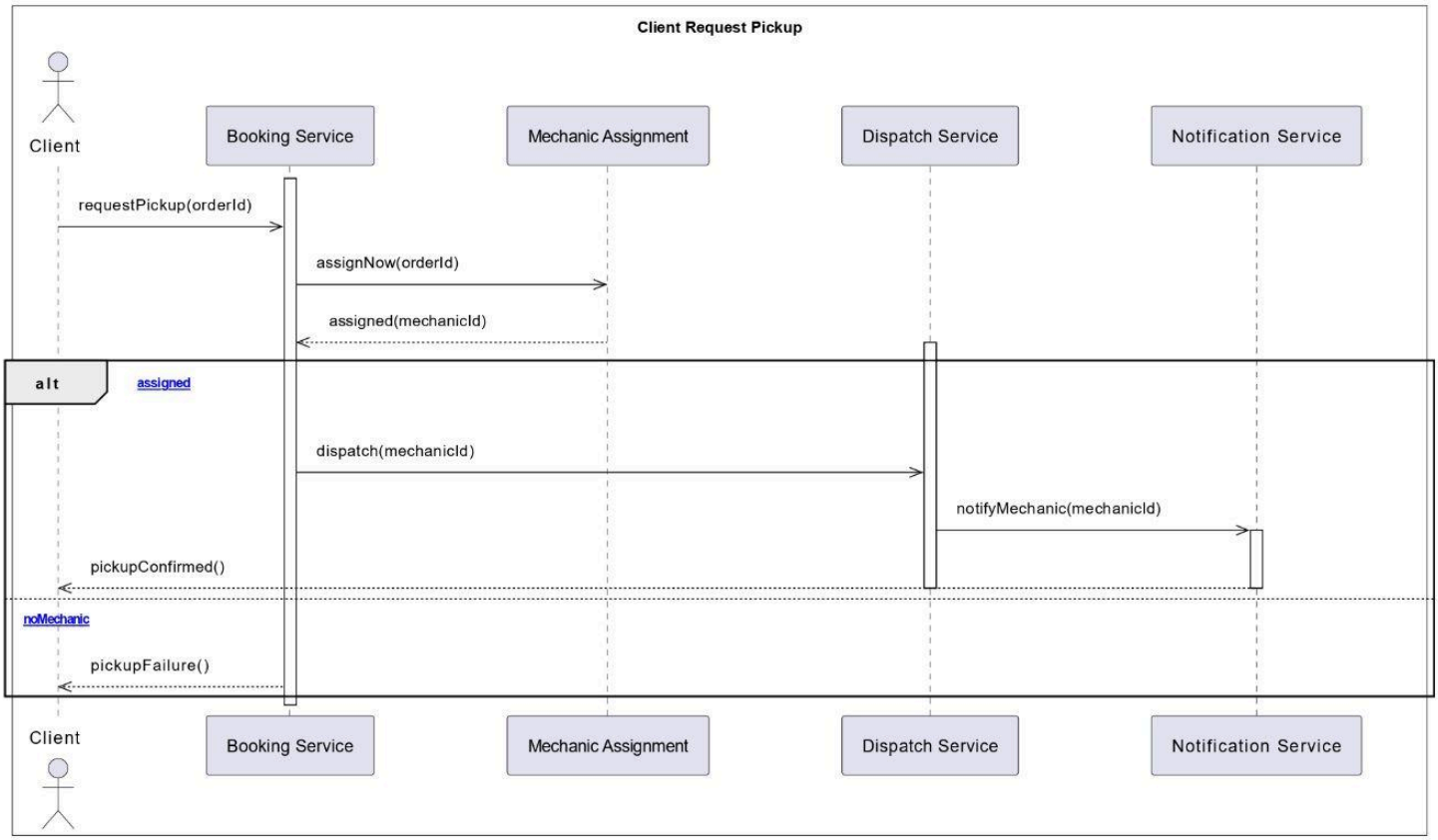

Client Update Personal Info
Client
User Profile Service
Database
updateProfile(data)
updateUser(data)
updateResult(success)
alt
success
showSuccess()
validationFail
showValidationError()
Client
User Profile Service
Database


Client Cancel Order
Client
Booking Service
Mechanic Assignment Service
Notification Service
cancelOrder(orderId)
checkStatus(orderId)
alt
status=pending
release(mechanicId)
sendCancelNotif(orderId)
cancelAck()
cancelReject()
Client
Booking Service
Mechanic Assignment Service
Notification Service

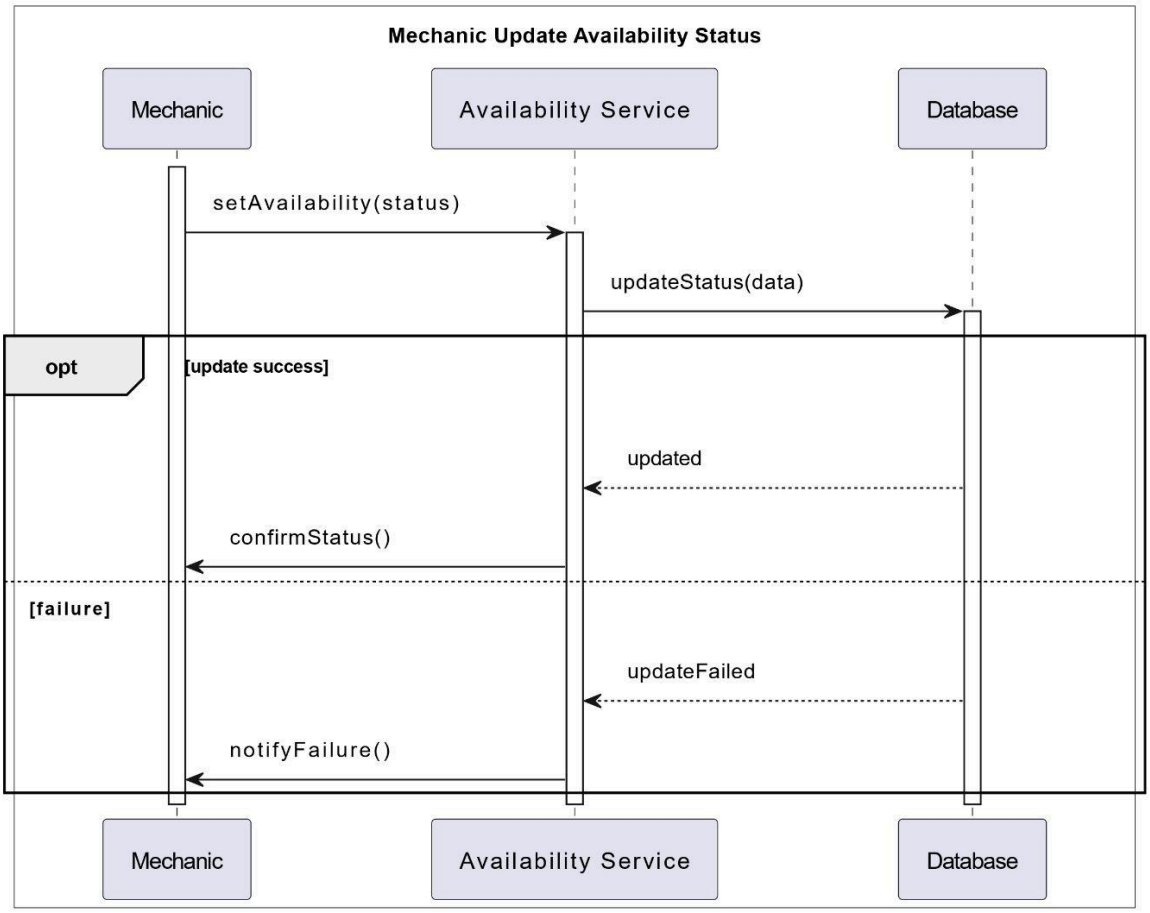

Client Share Location
Client
Geo Service
Database
shareLocation(lat, long)
saveLocation(data)
saved
alt
saved?
confirmSaved()
error
showError()
Client
Geo Service
Database

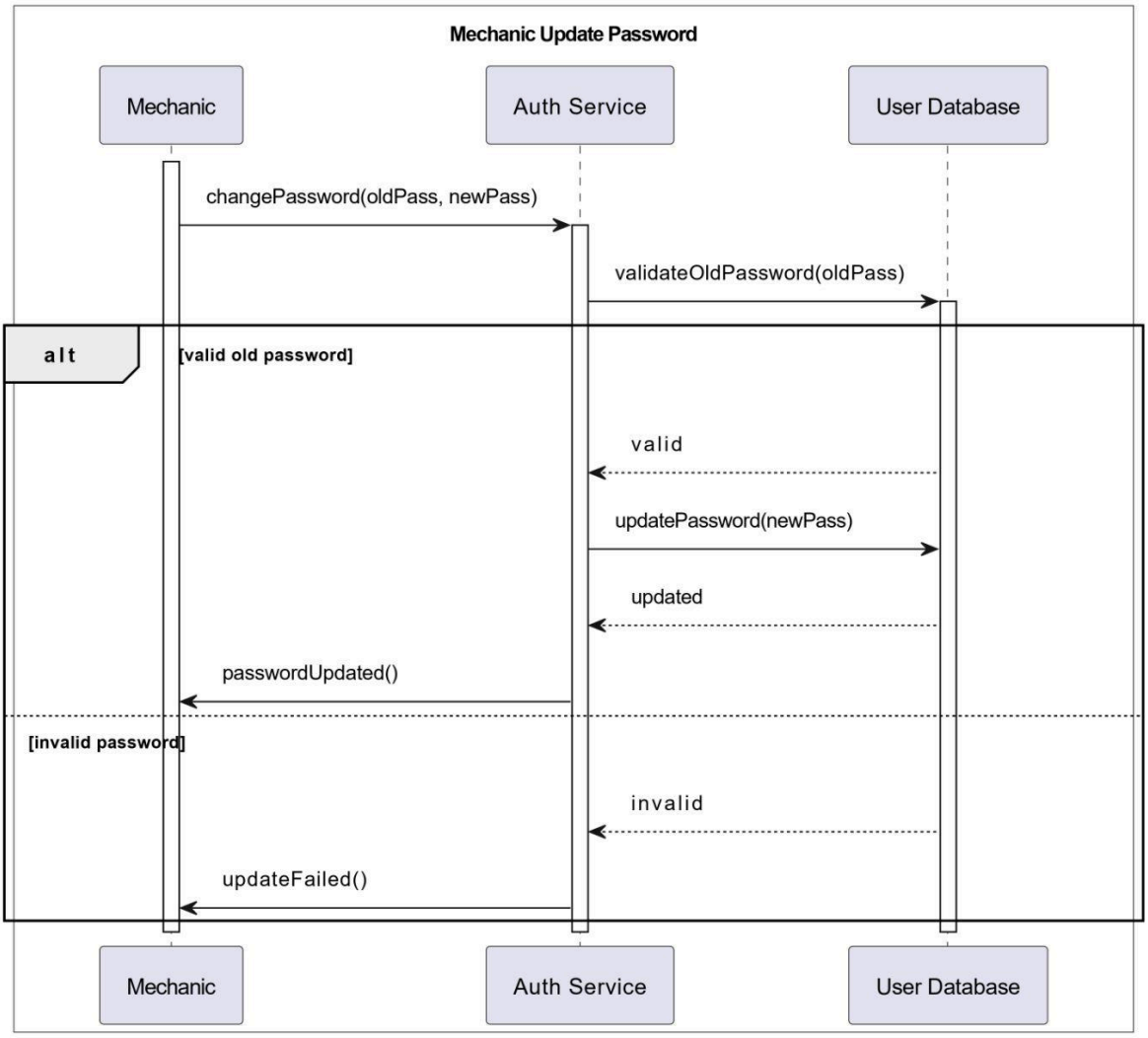

Client Search Nearest Mechanic
Client
Geo Service
Mechanic Registry
findNearest(radius)
filterByDistance(clientLoc, radius)
nearbyList
displayNearest(nearbyList)
loop
each mechanic
selectMechanic(mechanicId)
Client
Geo Service
Mechanic Registry

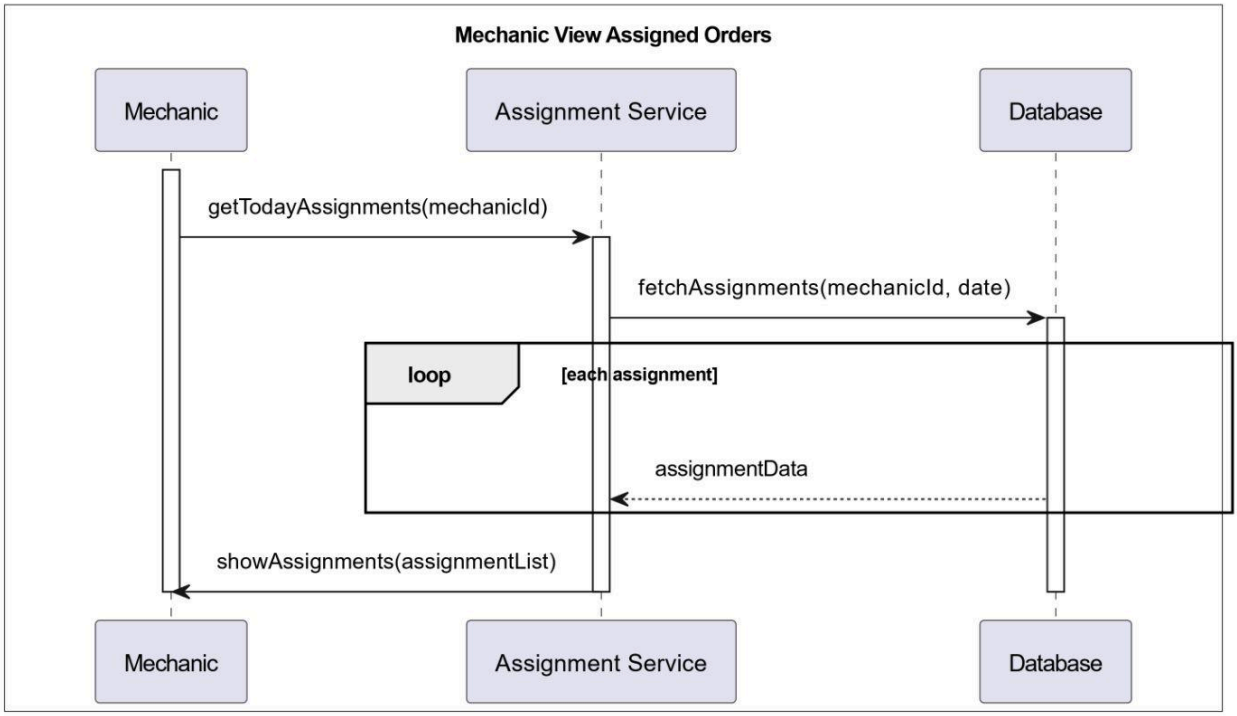

Client View Details & Directions
Client
Mechanic Registry
Directions API
getDetails(mechanicId)
details
getRoute(clientLoc, mechanicLoc)
alt
available
routeSteps
route
displayNoRouteMessage()
Client
Mechanic Registry
Directions API

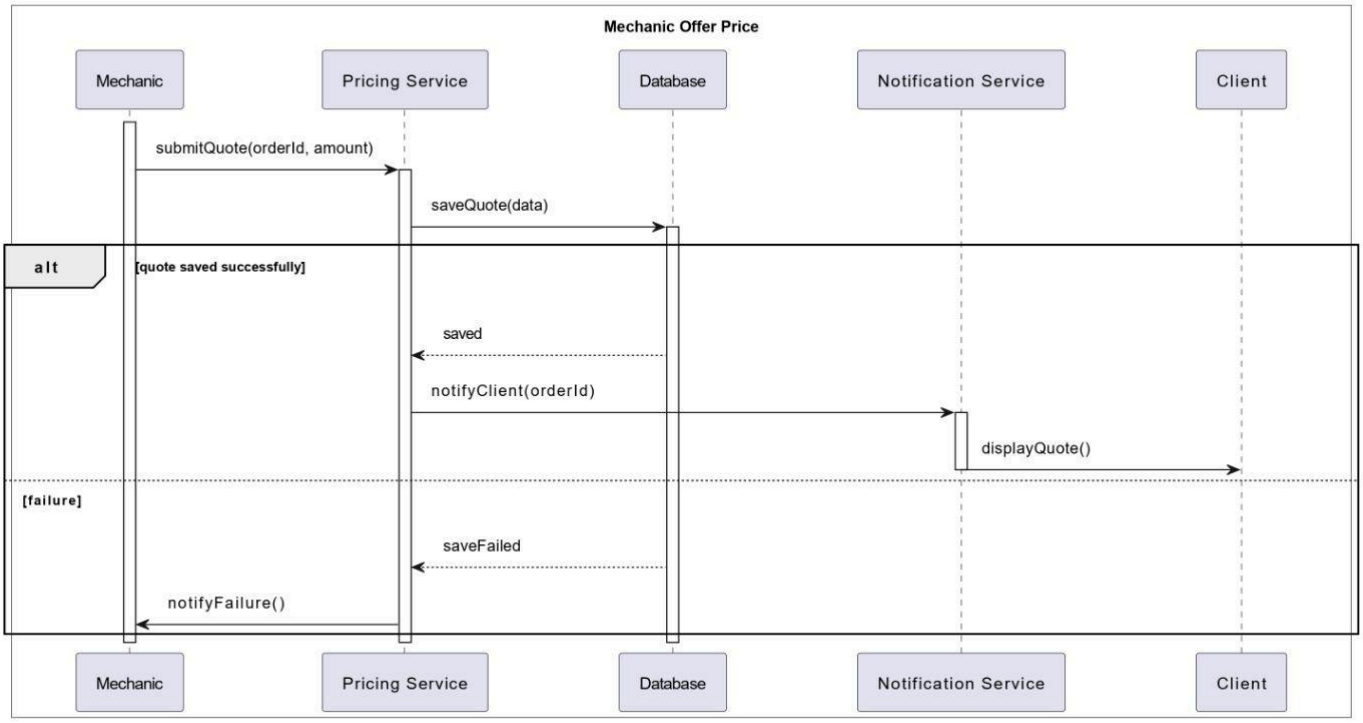

Client Request Pickup
Client
Booking Service
Mechanic Assignment
Dispatch Service
Notification Service
requestPickup(orderId)
assignNow(orderId)
assigned(mechanicId)
alt
assigned
dispatch(mechanicId)
notifyMechanic(mechanicId)
pickupConfirmed()
noMechanic
pickupFailure()
Client
Booking Service
Mechanic Assignment
Dispatch Service
Notification Service

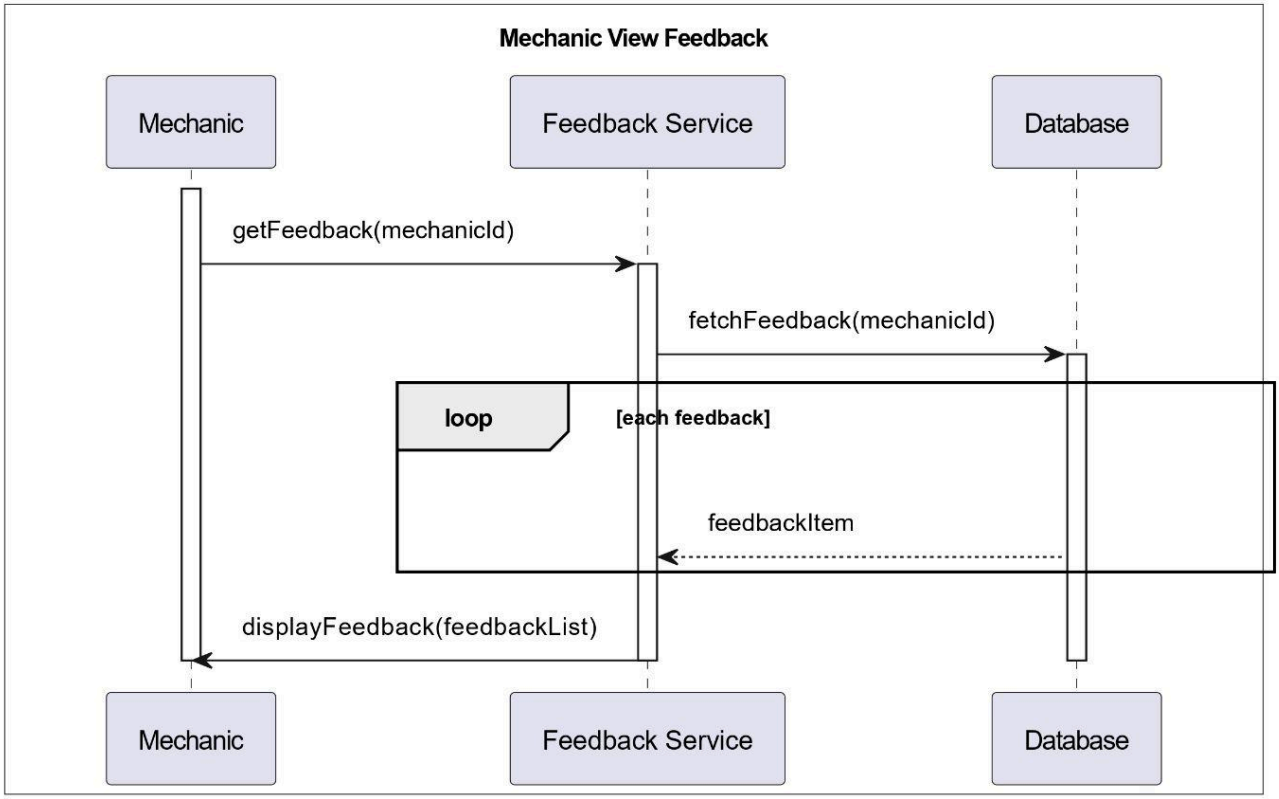
Mechanic Sequence Diagrams

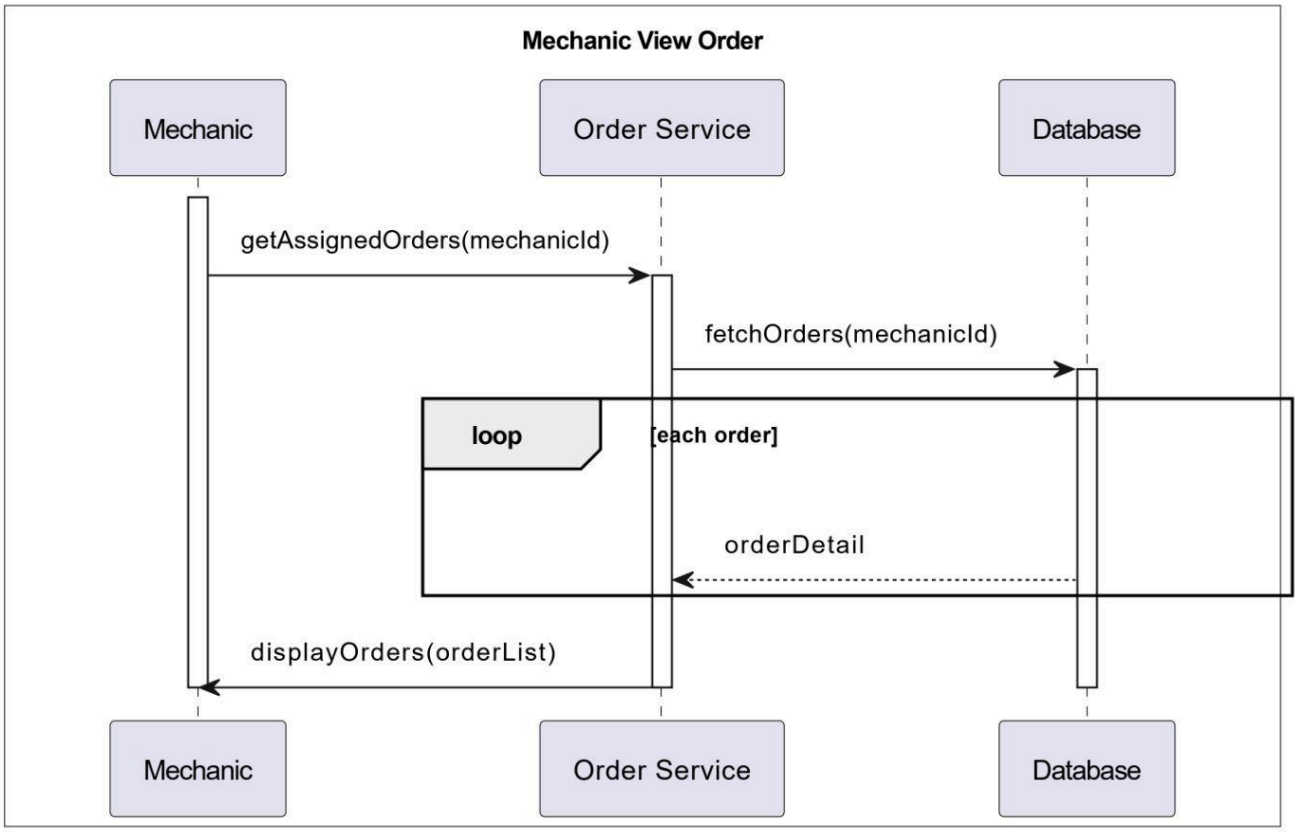


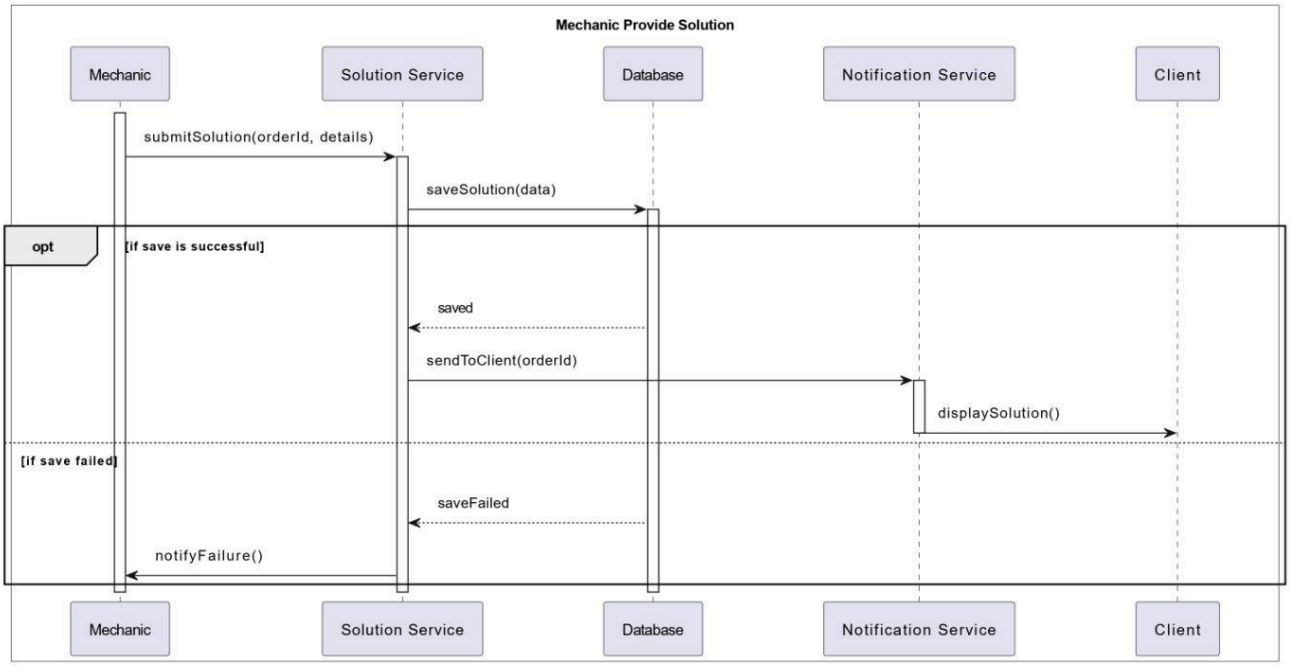

Mechanic Update Password
Mechanic
Auth Service
User Database
changePassword(oldPass, newPass)
validateOldPassword(oldPass)
alt
[valid old password]
valid
updatePassword(newPass)
updated
passwordUpdated()
[invalid password]
invalid
updateFailed()
Mechanic
Auth Service
User Database

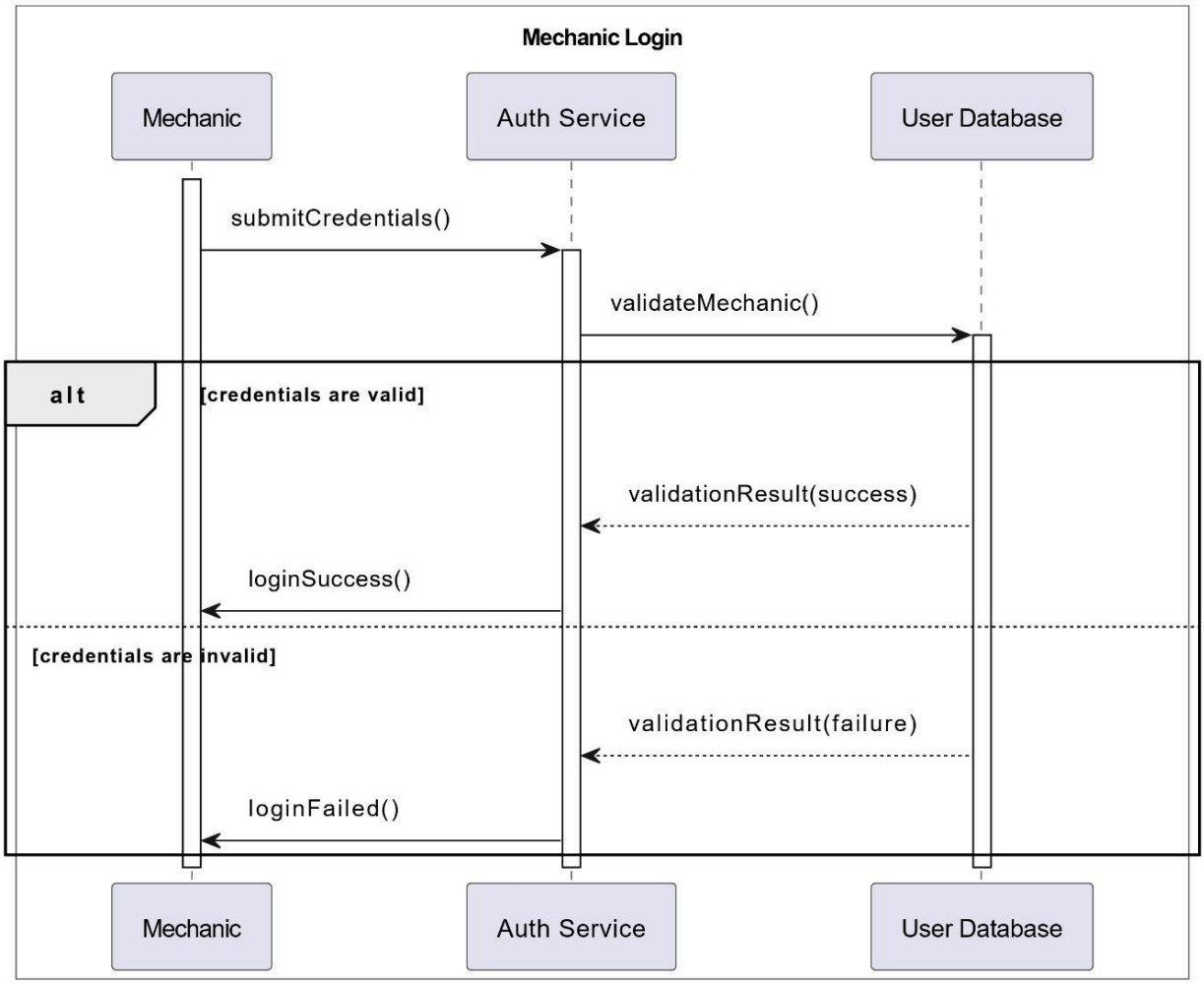

Mechanic View Assigned Orders
Mechanic
Assignment Service
Database
getTodayAssignments(mechanicId)
fetchAssignments(mechanicId, date)
loop
[each assignment]
assignmentData
showAssignments(assignmentList)
Mechanic
Assignment Service
Database

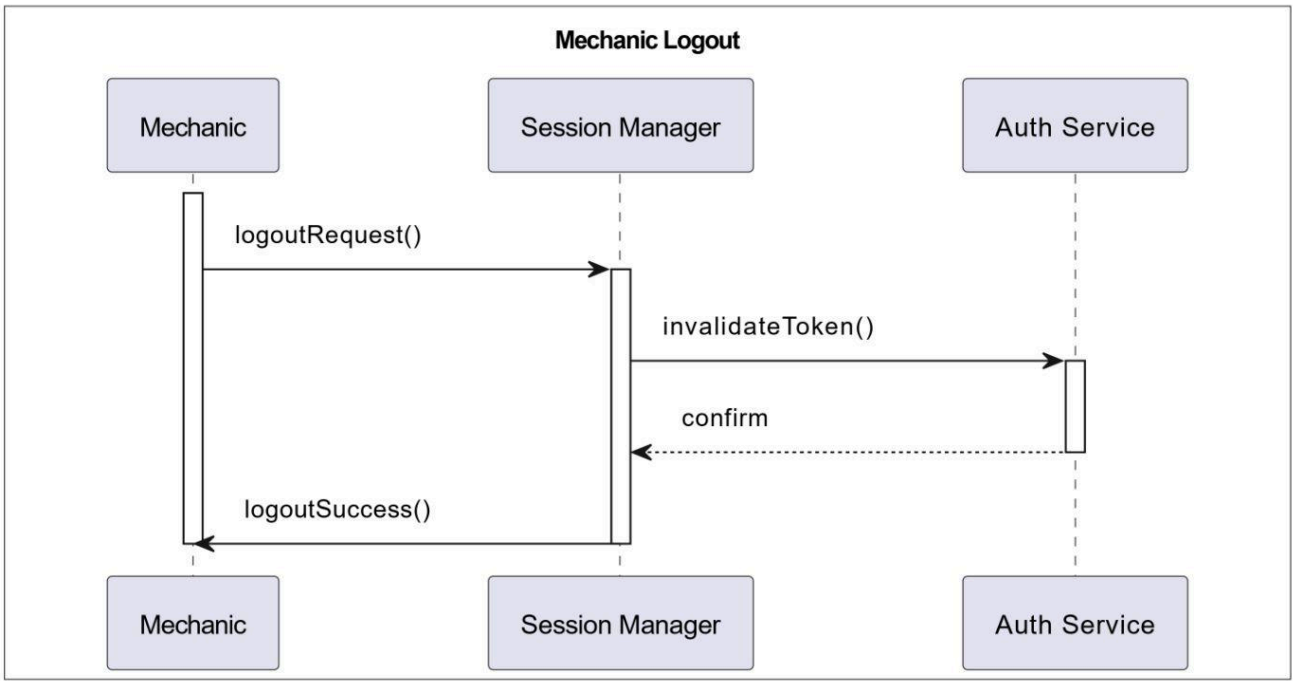

Mechanic Offer Price
Mechanic
Pricing Service
Database
Notification Service
Client
submitQuote(orderId, amount)
saveQuote(data)
alt
[quote saved successfully]
saved
notifyClient(orderId)
displayQuote()
[failure]
saveFailed
notifyFailure()
Mechanic
Pricing Service
Database
Notification Service
Client

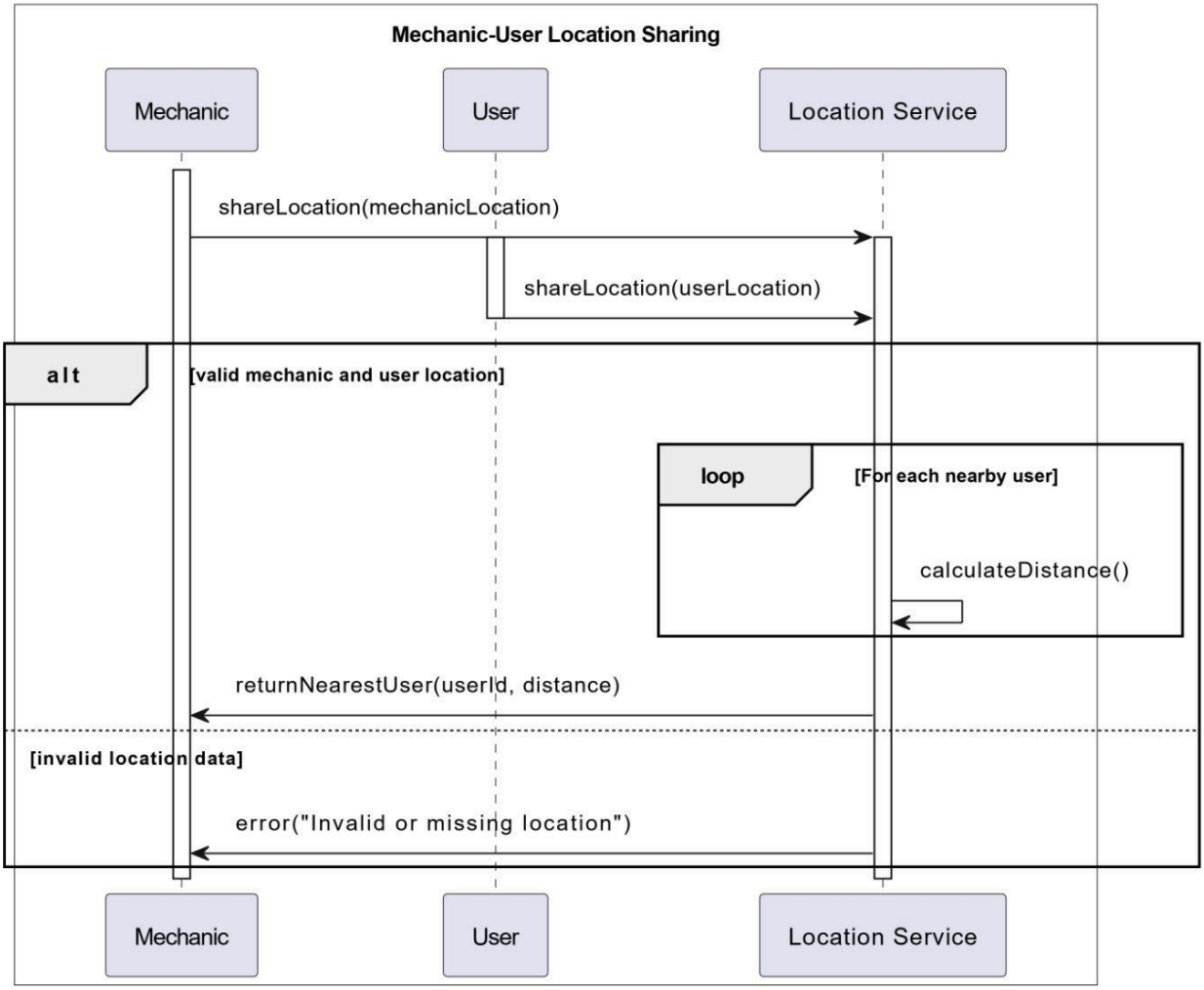

Mechanic View Feedback
Mechanic
Feedback Service
Database
getFeedback(mechanicId)
fetchFeedback(mechanicId)
loop
[each feedback]
feedbackItem
displayFeedback(feedbackList)
Mechanic
Feedback Service
Database


Mechanic View Order
Mechanic
Order Service
Database
getAssignedOrders(mechanicId)
fetchOrders(mechanicId)
loop
[each order]
orderDetail
displayOrders(orderList)
Mechanic
Order Service
Database

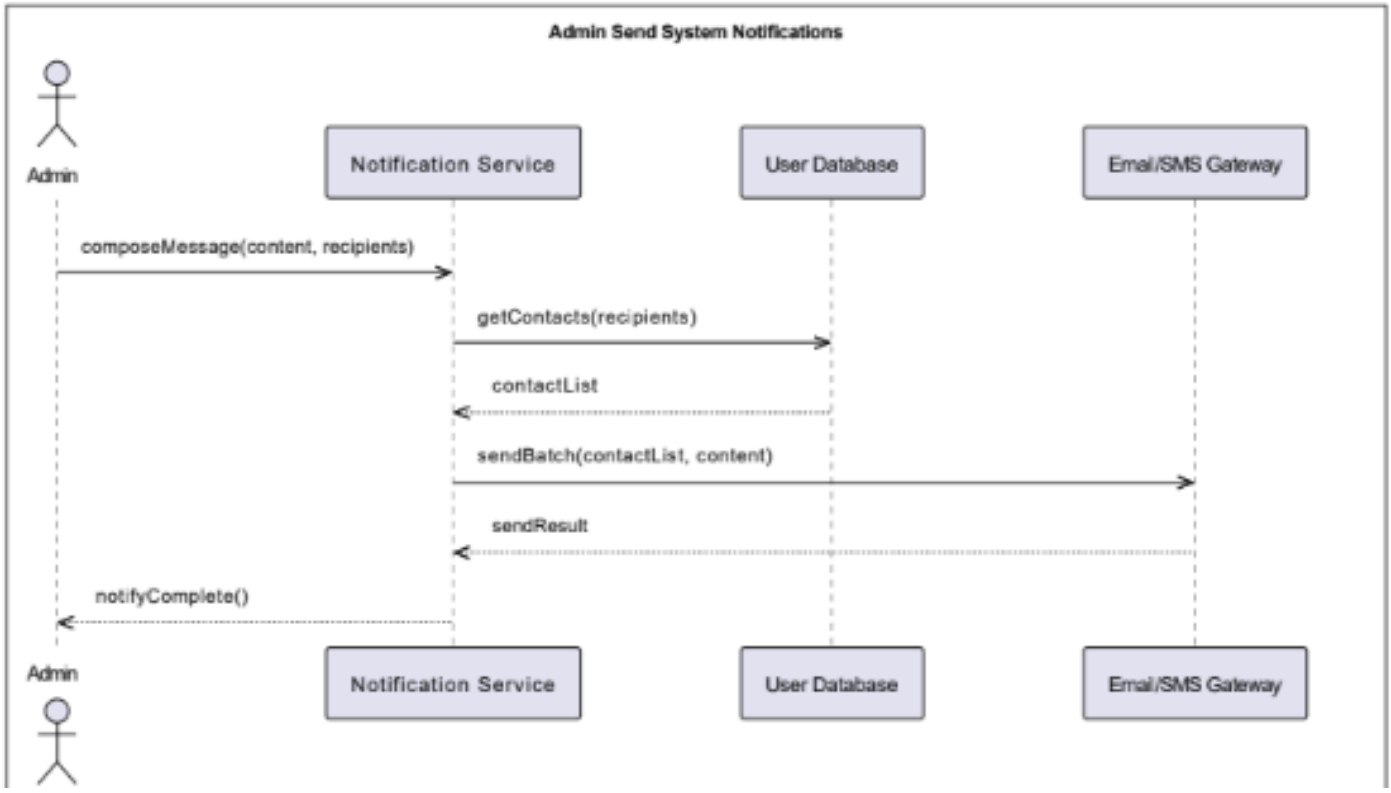

Mechanic Provide Solution
Mechanic
Solution Service
Database
Notification Service
Client
submitSolution(orderId, details)
saveSolution(data)
opt
[if save is successful]
saved
sendToClient(orderId)
displaySolution()
[if save failed]
saveFailed
notifyFailure()
Mechanic
Solution Service
Database
Notification Service
Client

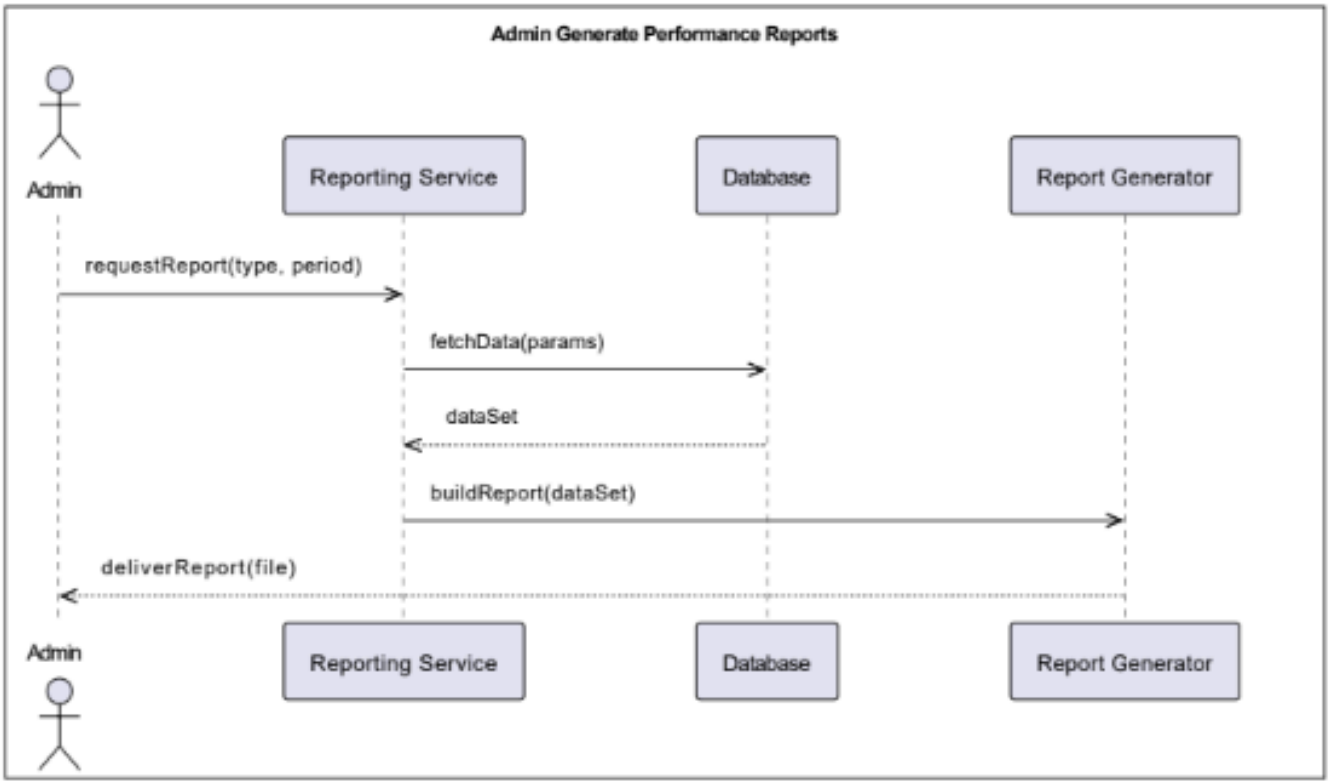

Mechanic Login
Mechanic
Auth Service
User Database
submitCredentials()
validateMechanic()
alt
[credentials are valid]
validationResult(success)
loginSuccess()
[credentials are invalid]
validationResult(failure)
loginFailed()
Mechanic
Auth Service
User Database

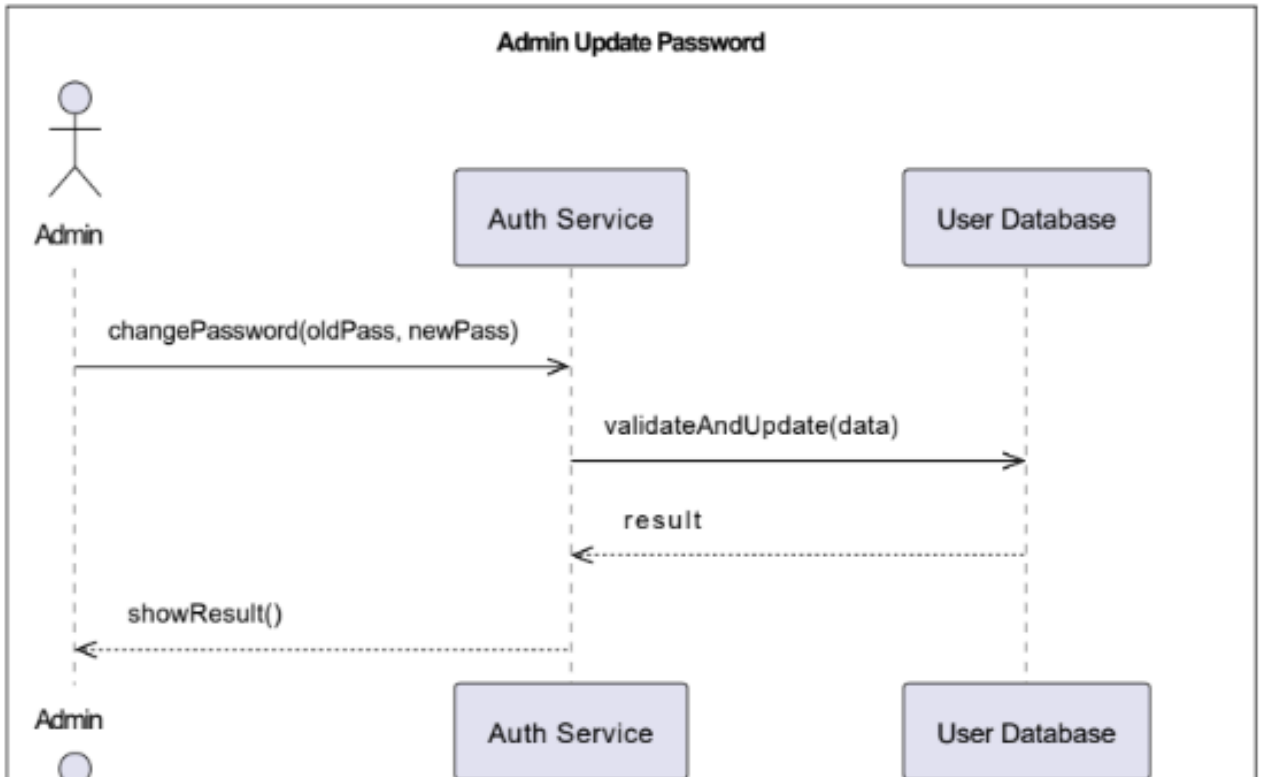

Mechanic Logout
Mechanic
Session Manager
Auth Service
logoutRequest()
invalidateToken()
confirm
logoutSuccess()
Mechanic
Session Manager
Auth Service

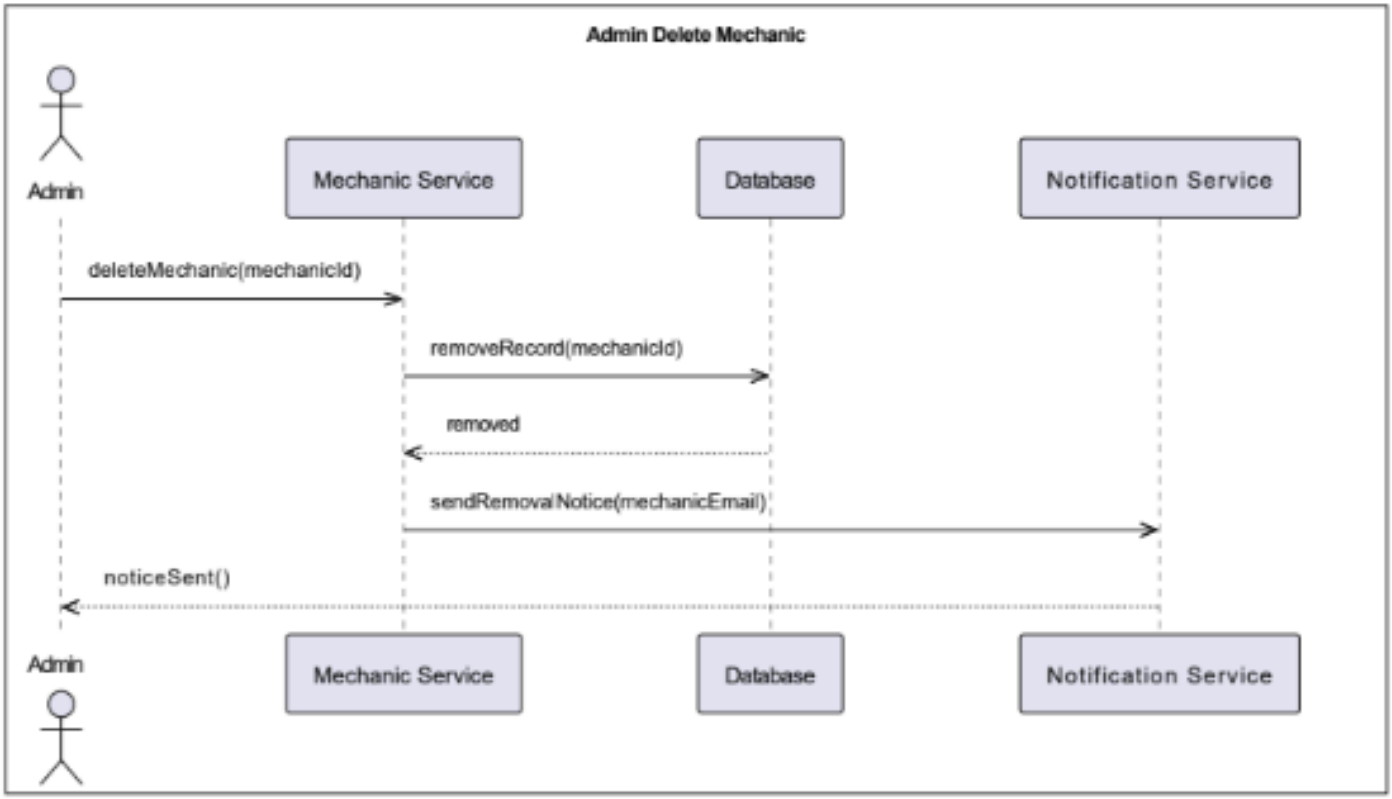

Mechanic-User Location Sharing
Mechanic
User
Location Service
shareLocation(mechanicLocation)
shareLocation(userLocation)
alt
[valid mechanic and user location]
loop
[For each nearby user]
calculateDistance()
returnNearestUser(userId, distance)
[invalid location data]
error("Invalid or missing location")
Mechanic
User
Location Service

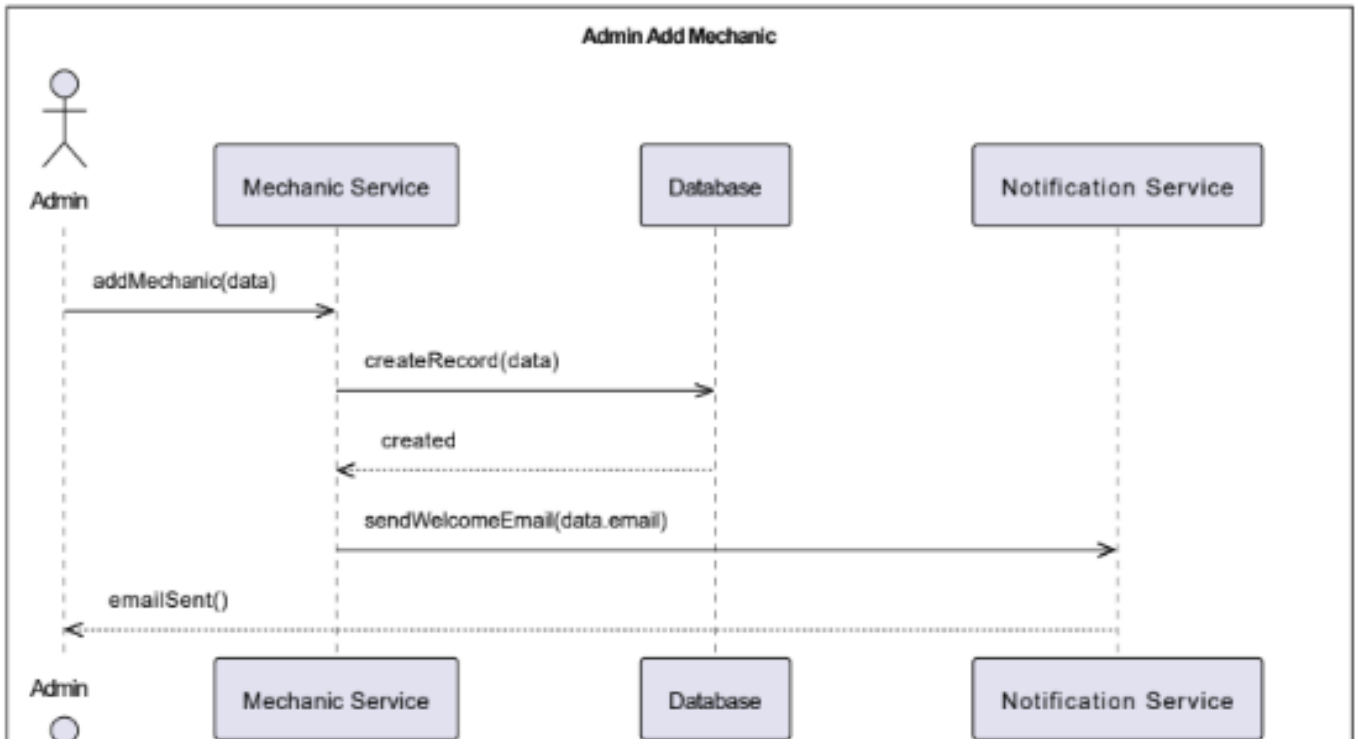
Admin Sequence Diagram

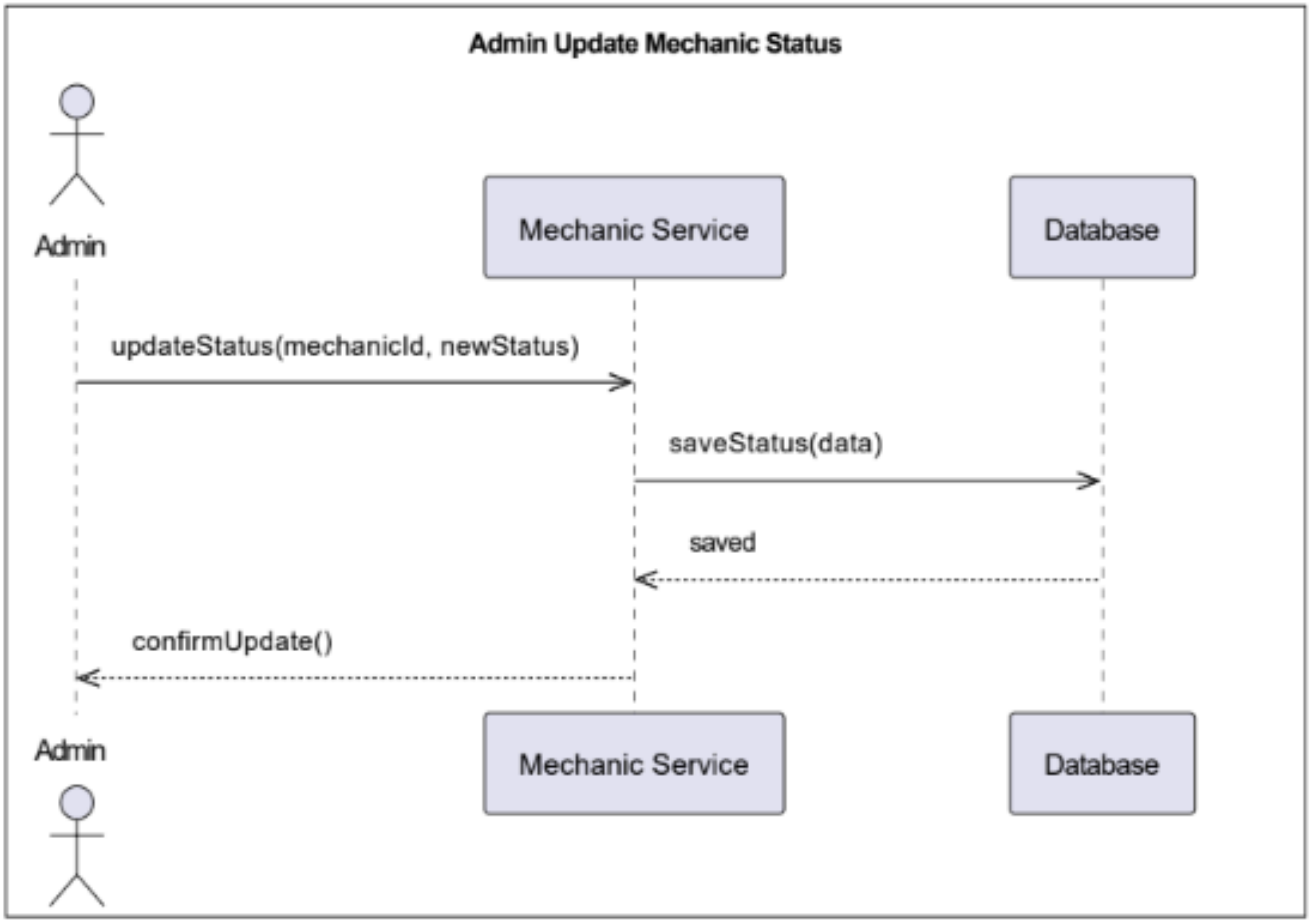


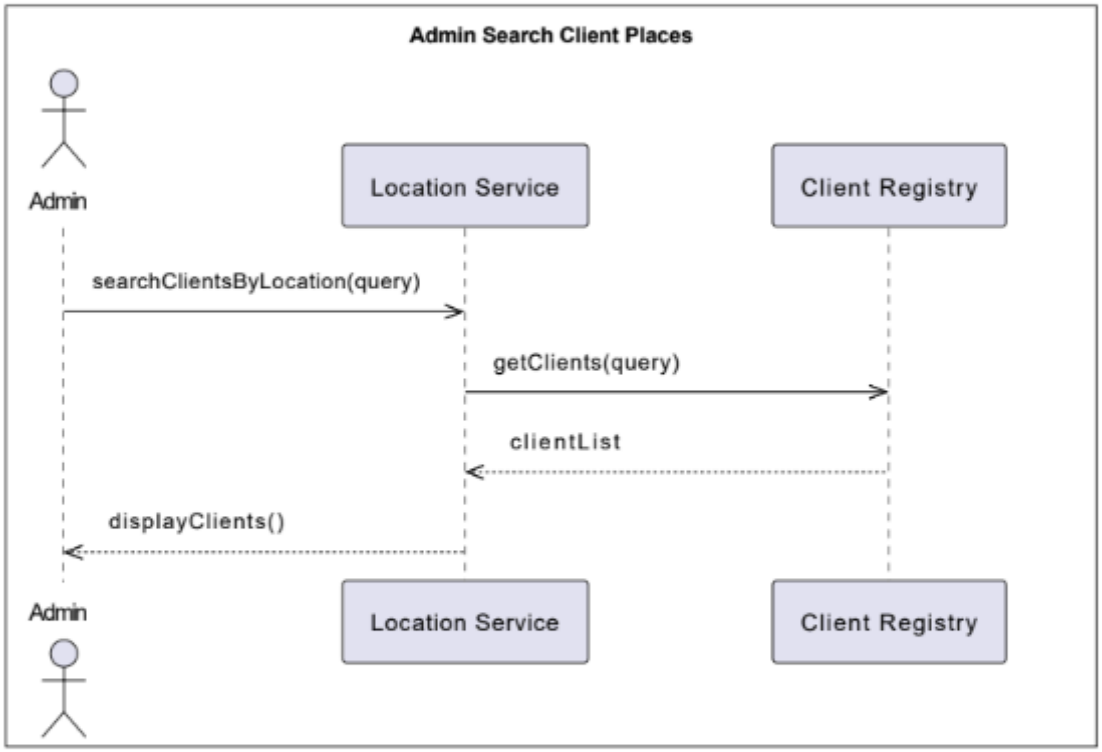


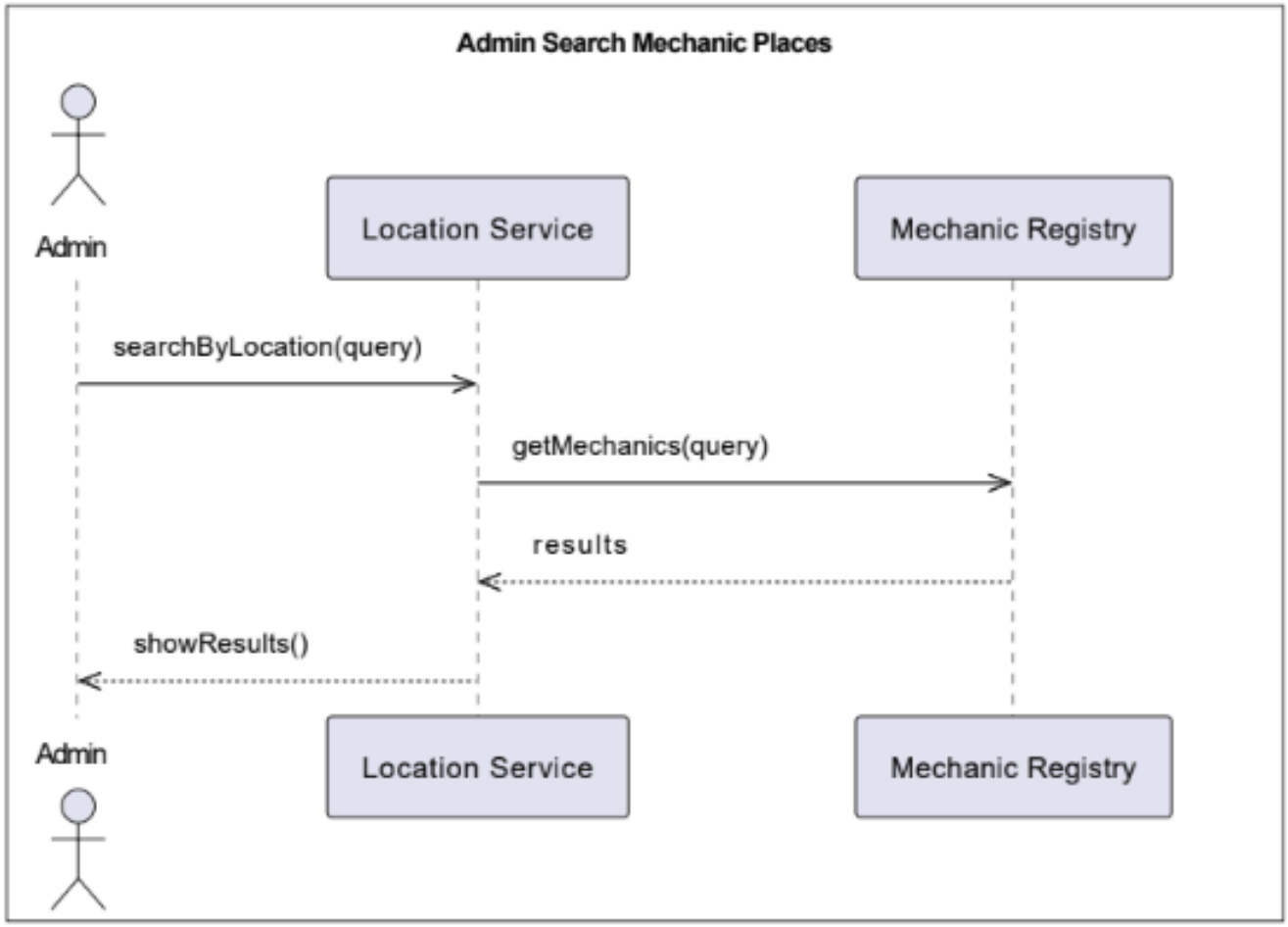

Admin Update Password
Admin
Auth Service
User Database
changePassword(oldPass, newPass)
validateAndUpdate(data)
result
showResult()
Admin
Auth Service
User Database

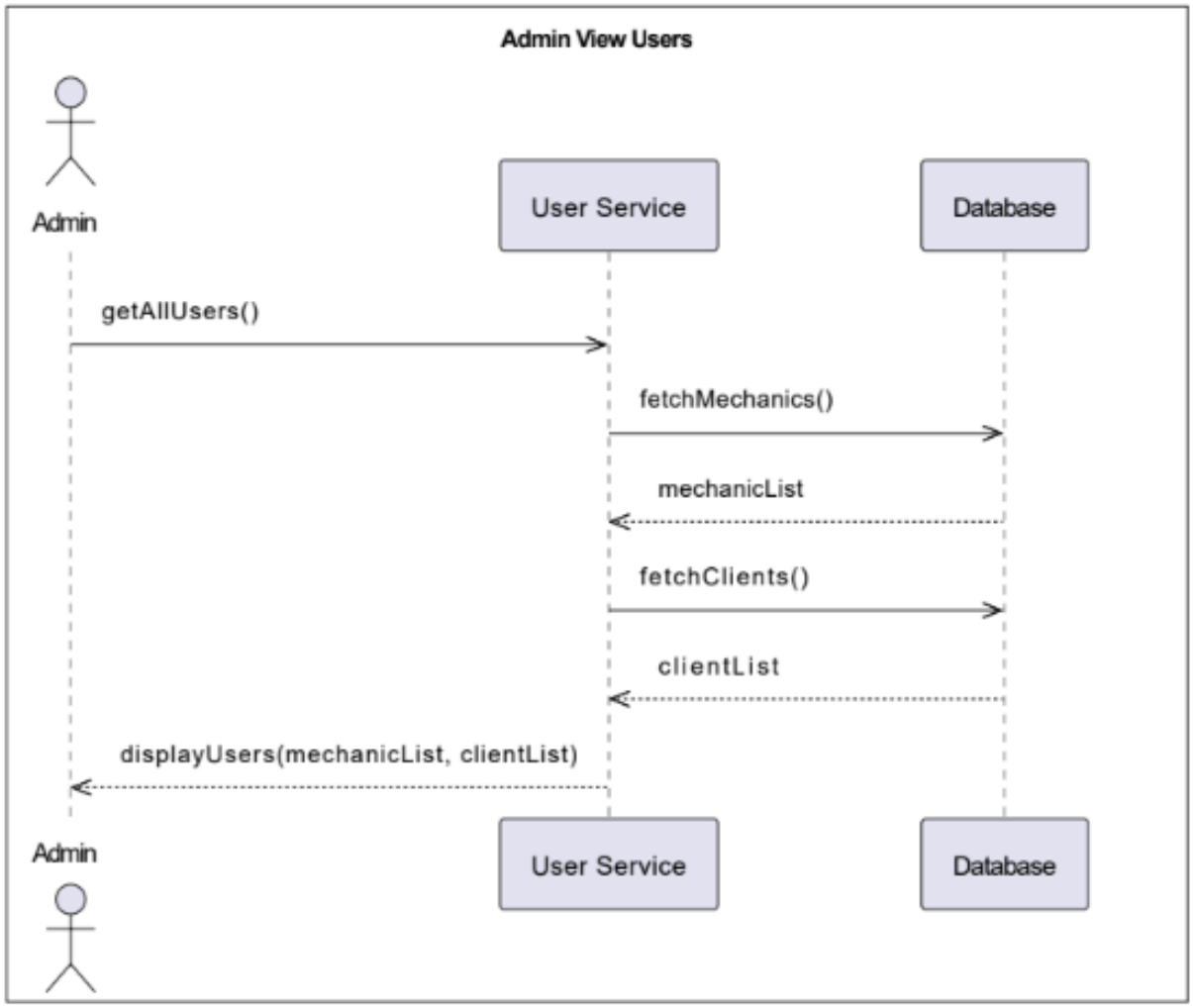

Admin Delete Mechanic
Admin
Mechanic Service
Database
Notification Service
deleteMechanic(mechanicId)
removeRecord(mechanicId)
removed
sendRemovalNotice(mechanicEmail)
noticeSent()
Admin
Mechanic Service
Database
Notification Service

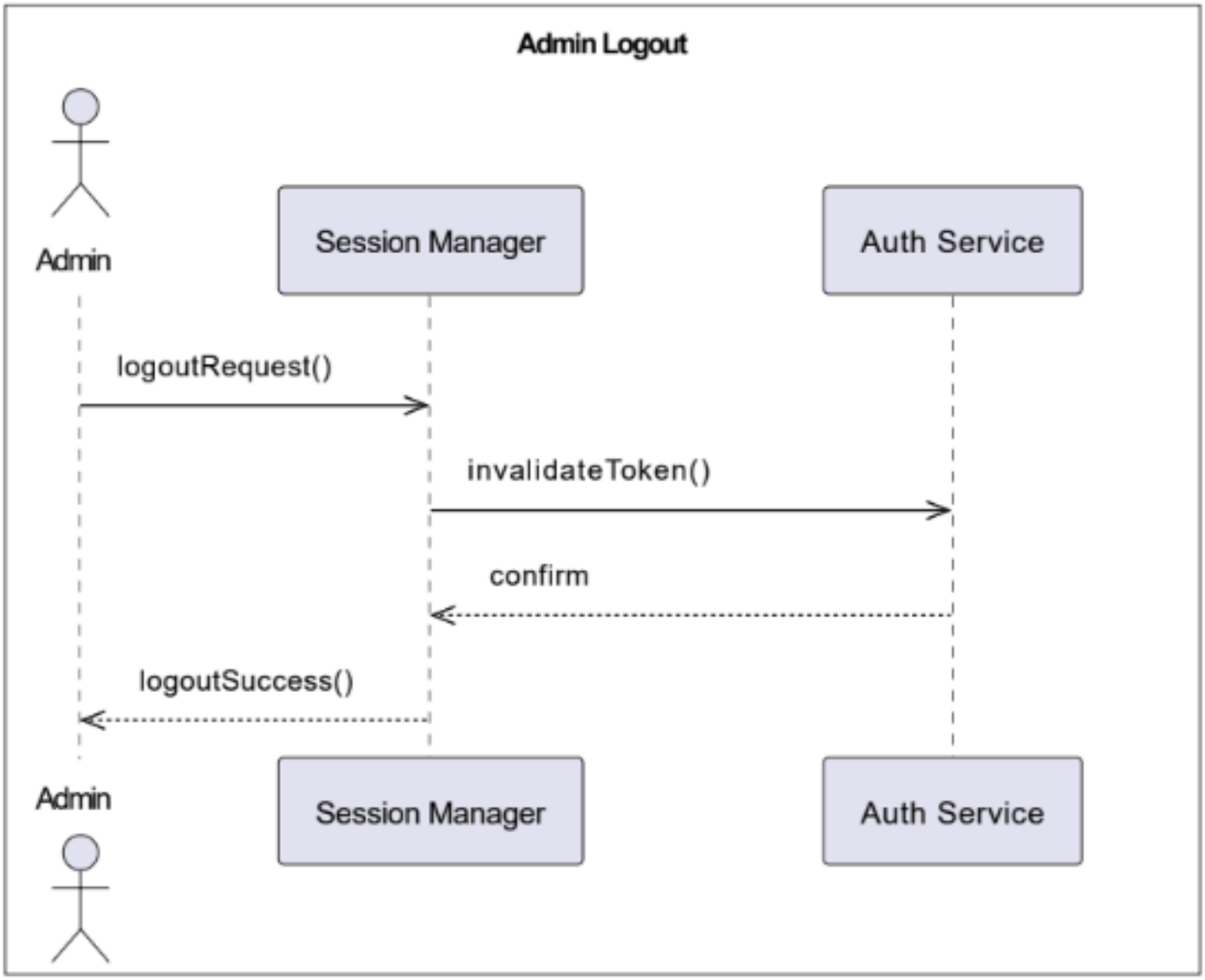

Admin Add Mechanic
Admin
Mechanic Service
Database
Notification Service
addMechanic(data)
createRecord(data)
created
sendWelcomeEmail(data.email)
emailSent()
Admin
Mechanic Service
Database
Notification Service

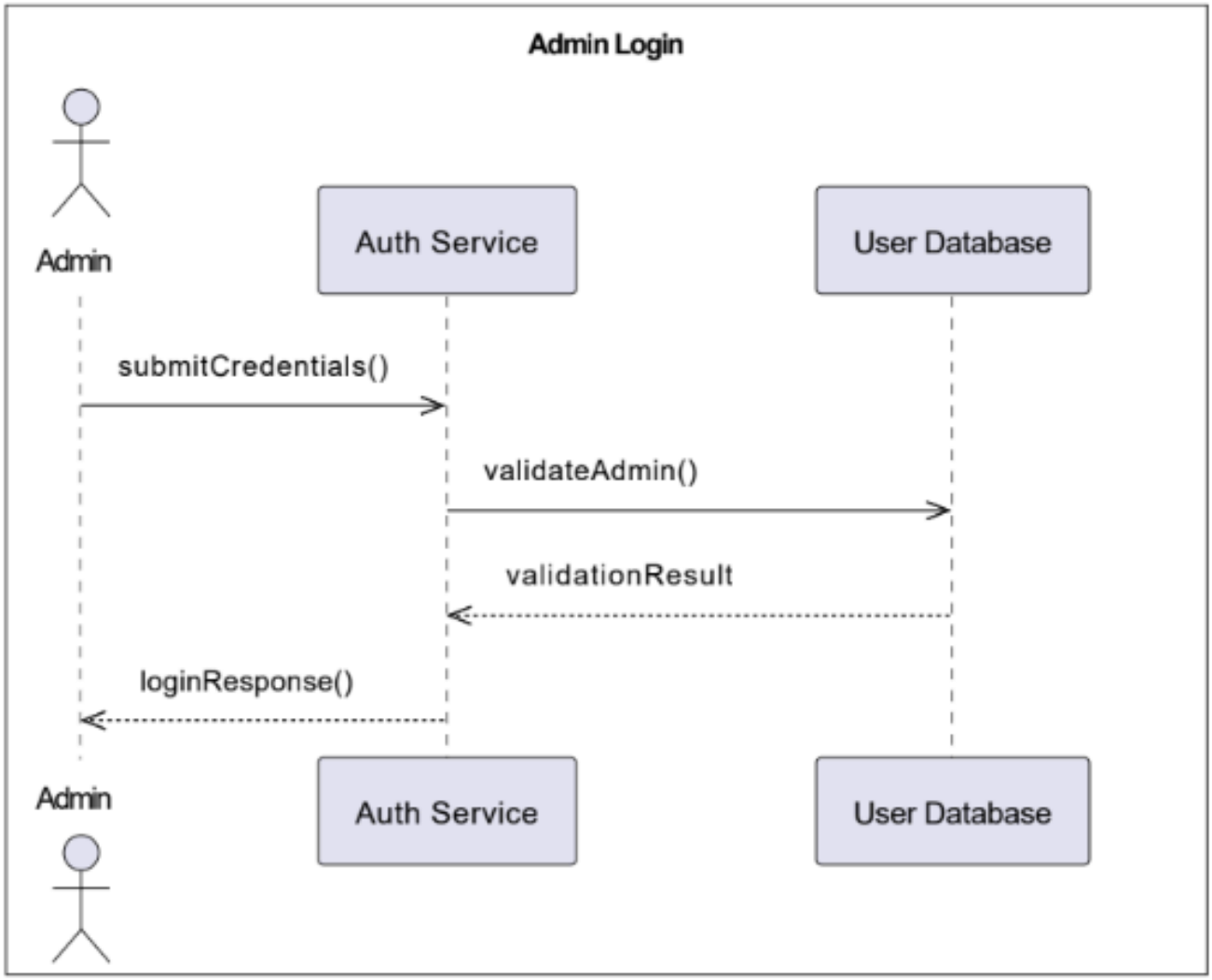

Admin Update Mechanic Status
Admin
Mechanic Service
Database
updateStatus(mechanicId, newStatus)
saveStatus(data)
saved
confirmUpdate()
Admin
Mechanic Service
Database


Admin Search Client Places
Admin
Location Service
Client Registry
searchClientsByLocation(query)
getClients(query)
clientList
displayClients()
Admin
Location Service
Client Registry

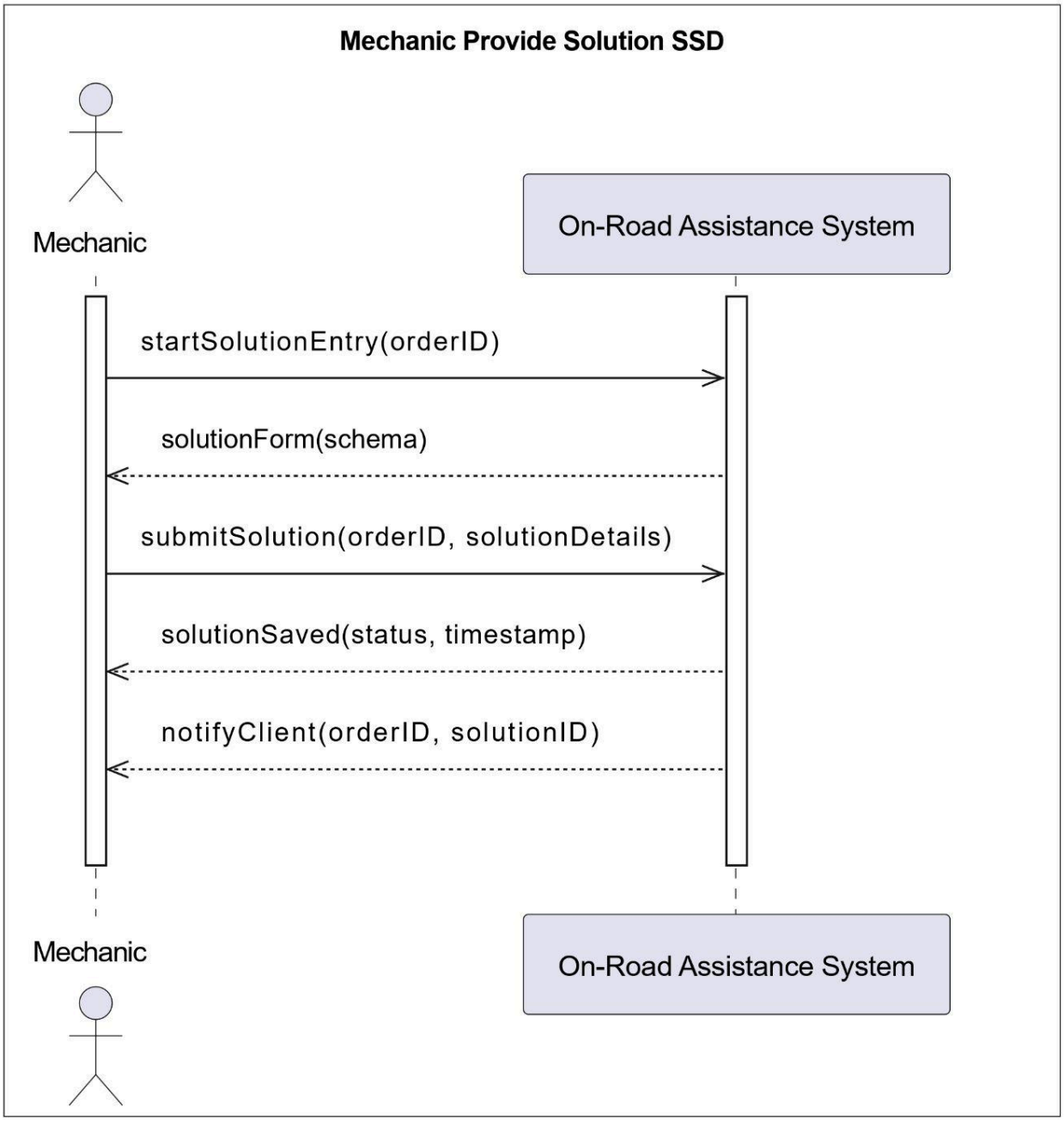

Admin Search Mechanic Places
Admin
Location Service
Mechanic Registry
searchByLocation(query)
getMechanics(query)
results
showResults()
Admin
Location Service
Mechanic Registry

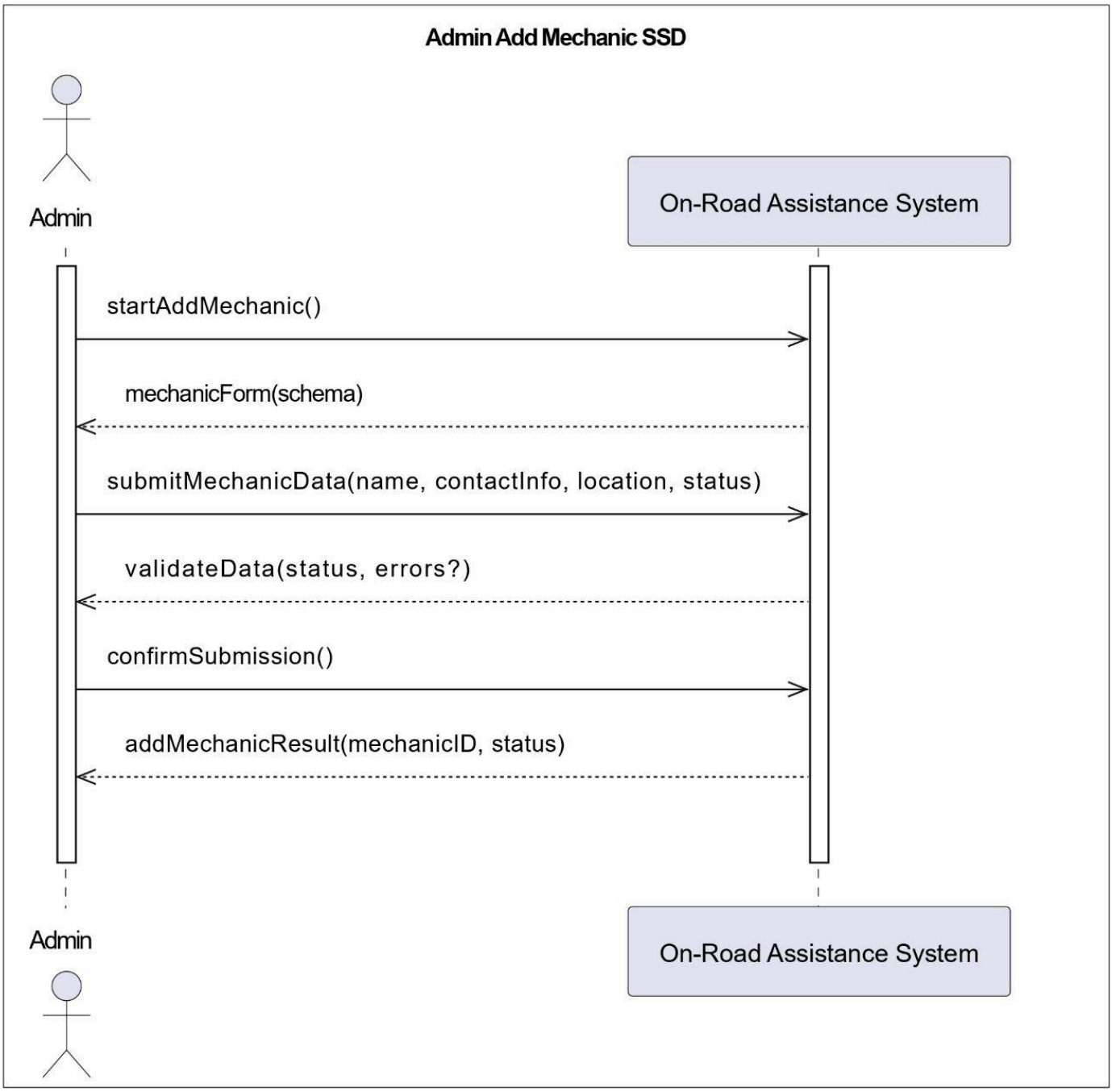

Admin View Users
Admin
User Service
Database
getAllUsers()
fetchMechanics()
mechanicList
fetchClients()
clientList
displayUsers(mechanicList, clientList)
Admin
User Service
Database

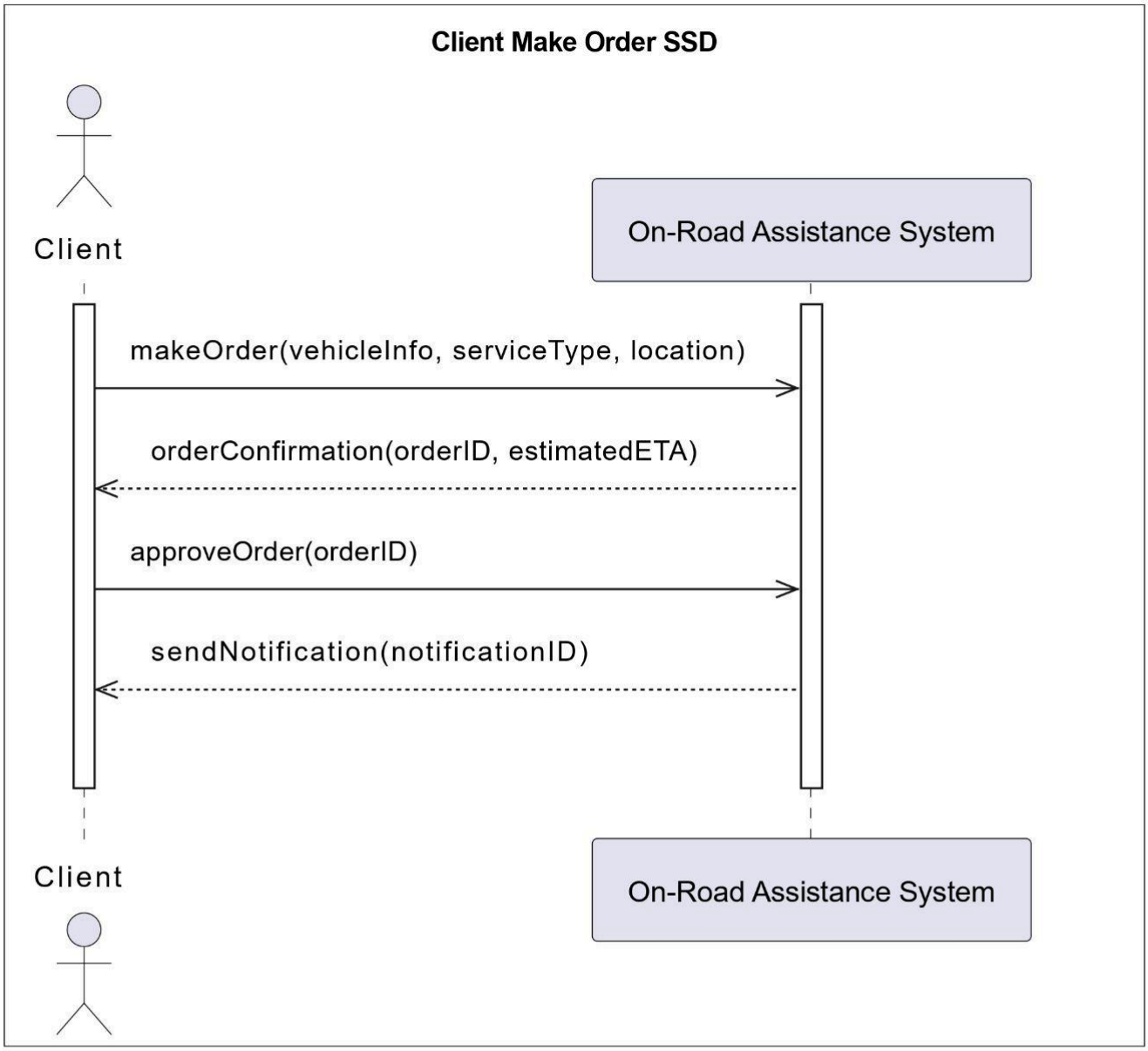

Admin Logout
Admin
Session Manager
Auth Service
logoutRequest()
invalidateToken()
confirm
logoutSuccess()
Admin
Session Manager
Auth Service

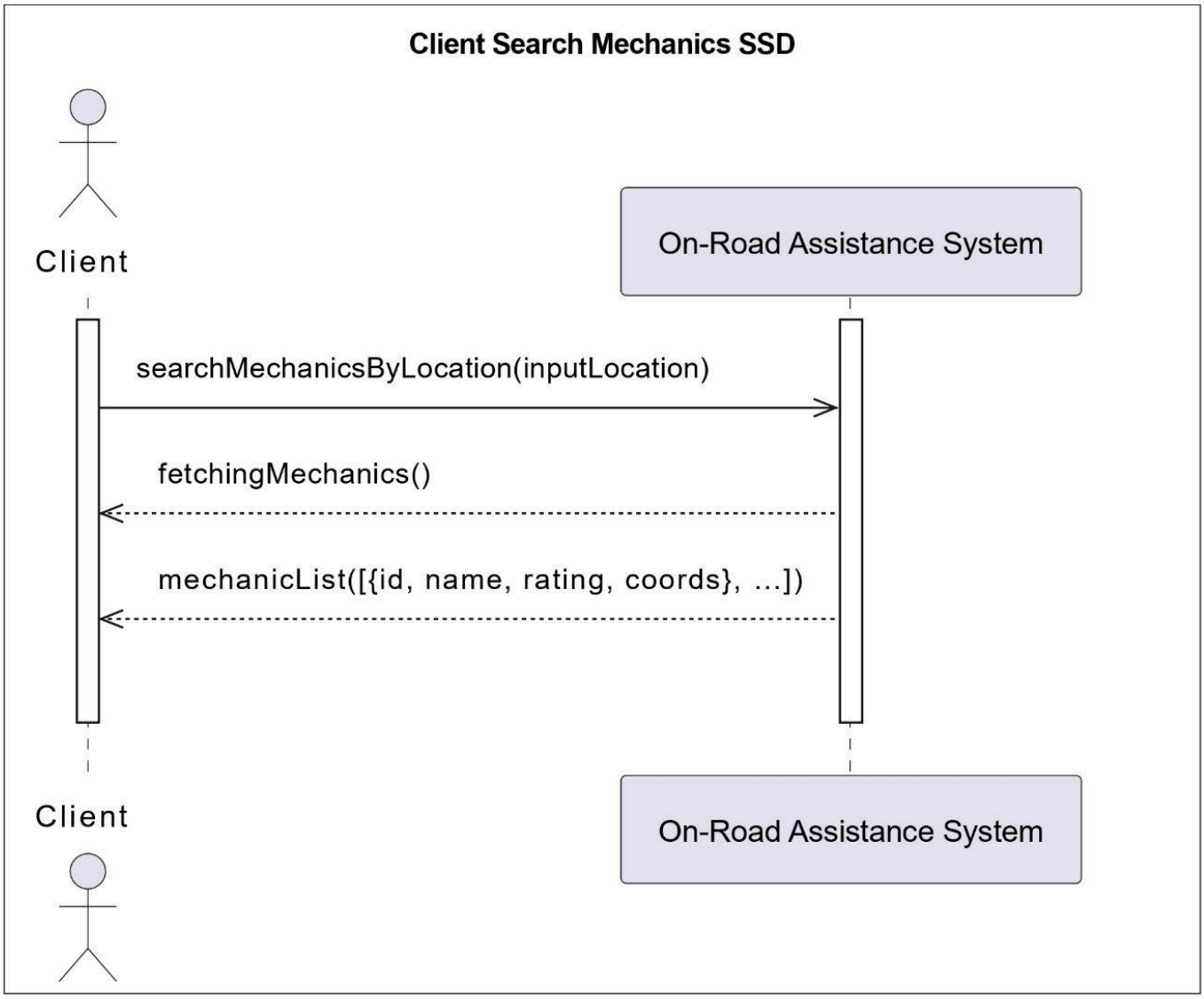

Admin Login
Admin
Auth Service
User Database
submitCredentials()
validateAdmin()
validationResult
loginResponse()
Admin
Auth Service
User Database

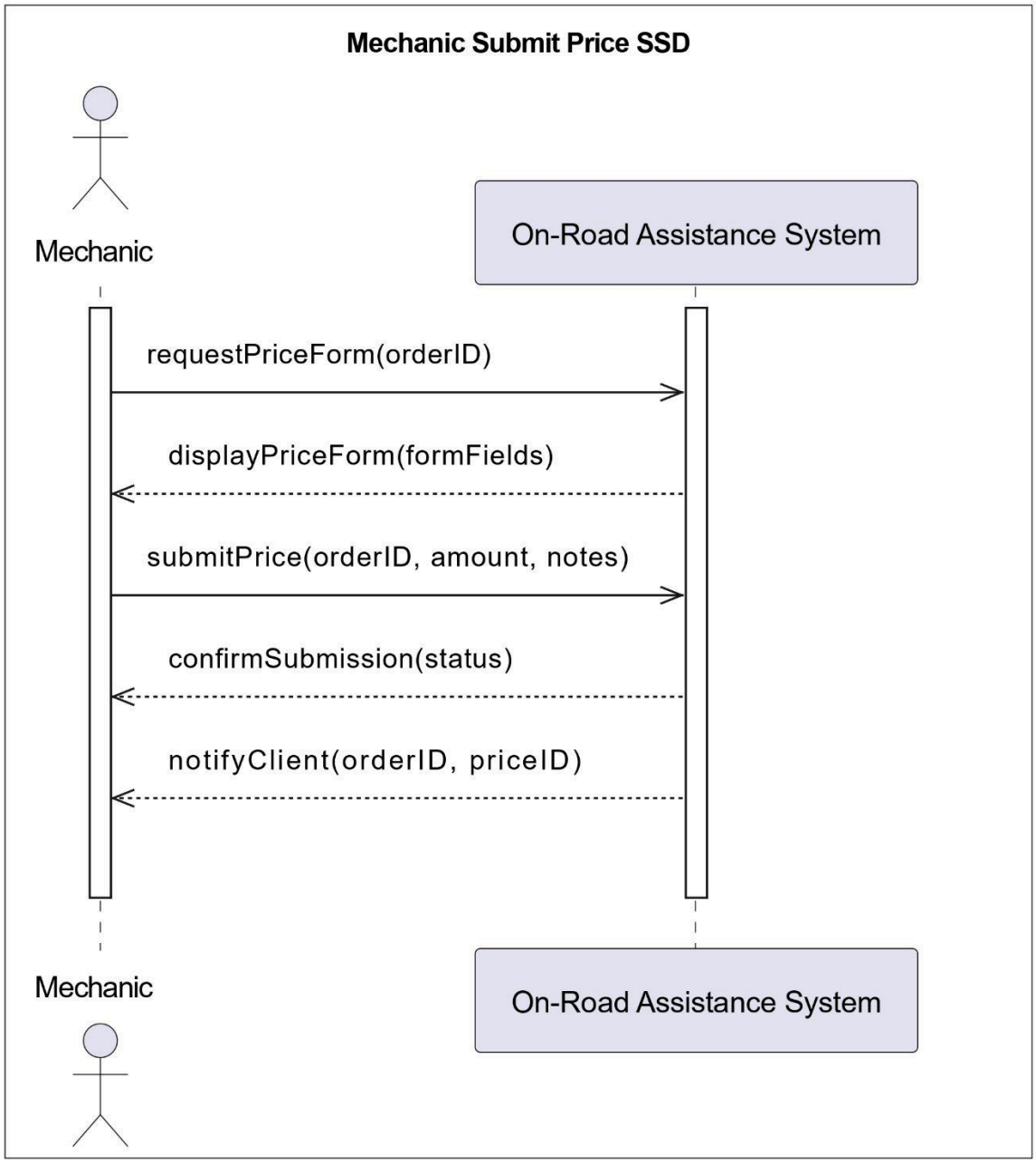
System Sequence Diagrams

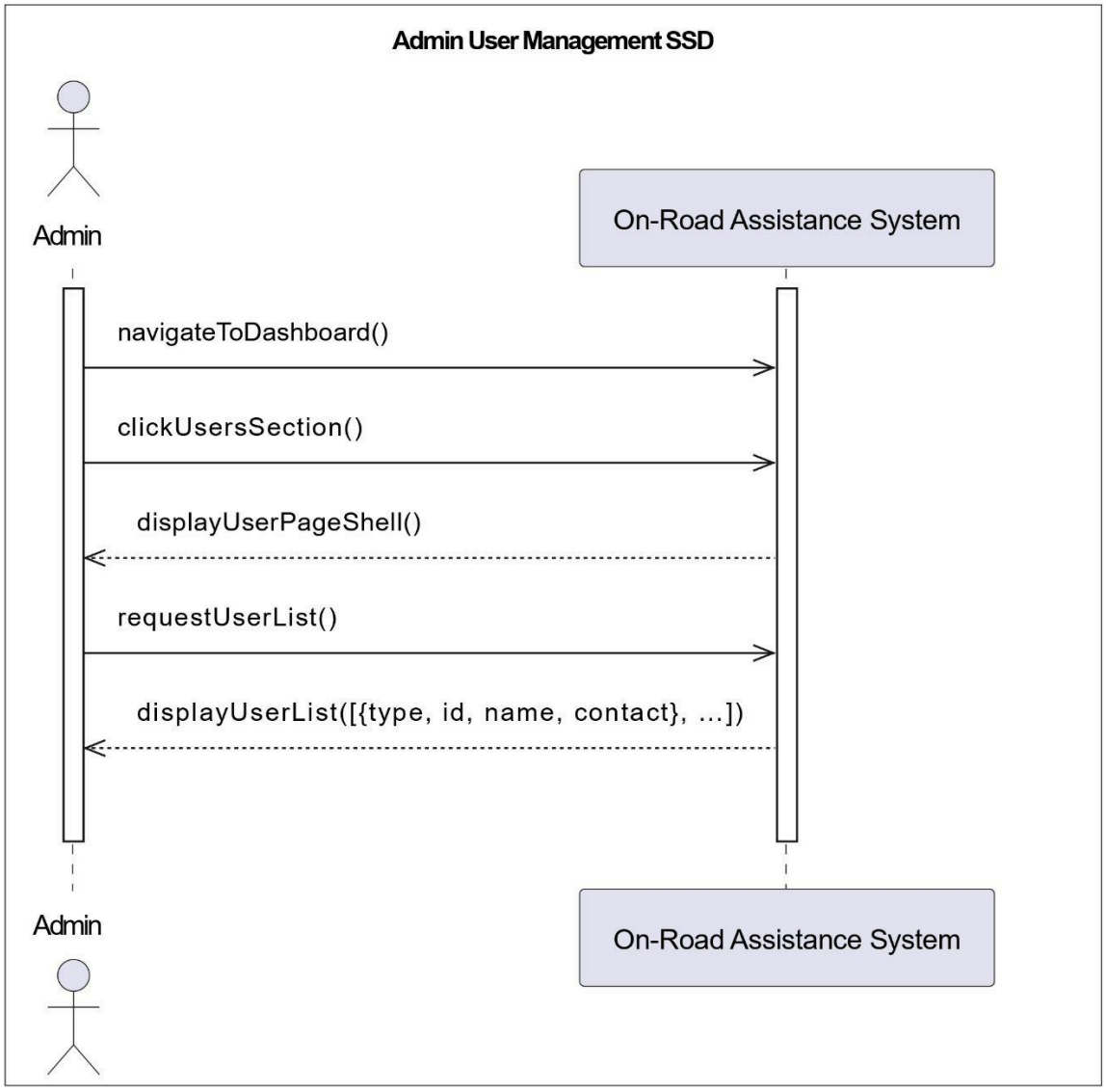



Admin Add Mechanic SSD
Admin
On-Road Assistance System
startAddMechanic()
mechanicForm(schema)
submitMechanicData(name, contactInfo, location, status)
validateData(status, errors?)
confirmSubmission()
addMechanicResult(mechanicID, status)
Admin
On-Road Assistance System


Client Make Order SSD
Client
On-Road Assistance System
makeOrder(vehicleInfo, serviceType, location)
orderConfirmation(orderID, estimatedETA)
approveOrder(orderID)
sendNotification(notificationID)
Client
On-Road Assistance System

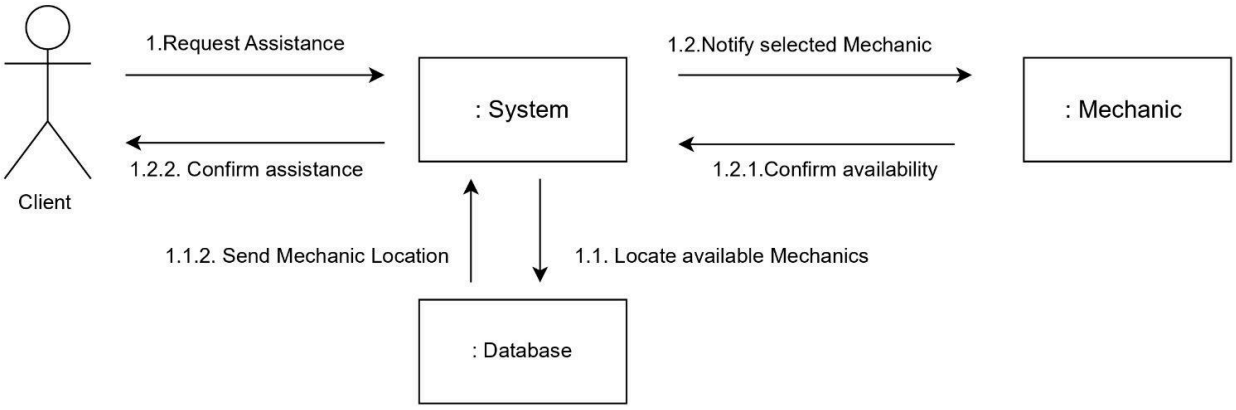

Client Search Mechanics SSD
Client
On-Road Assistance System
searchMechanicsByLocation(inputLocation)
fetchingMechanics()
mechanicList([{id, name, rating, coords}, ...])
Client
On-Road Assistance System


Mechanic Submit Price SSD
Mechanic
On-Road Assistance System
requestPriceForm(orderID)
displayPriceForm(formFields)
submitPrice(orderID, amount, notes)
confirmSubmission(status)
notifyClient(orderID, priceID)
Mechanic
On-Road Assistance System

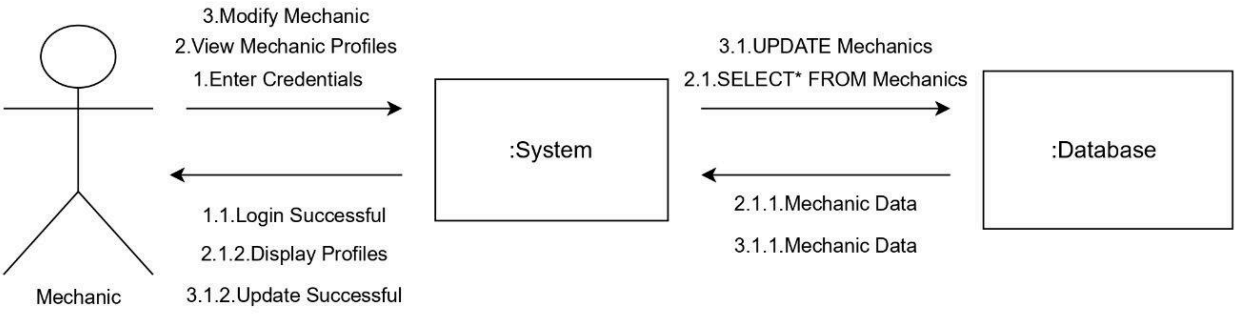

Admin User Management SSD
Admin
On-Road Assistance System
navigateToDashboard()
clickUsersSection()
displayUserPageShell()
requestUserList()
displayUserList([{type, id, name, contact}, ...])
Admin
On-Road Assistance System

Collaboration/Communication Diagrams

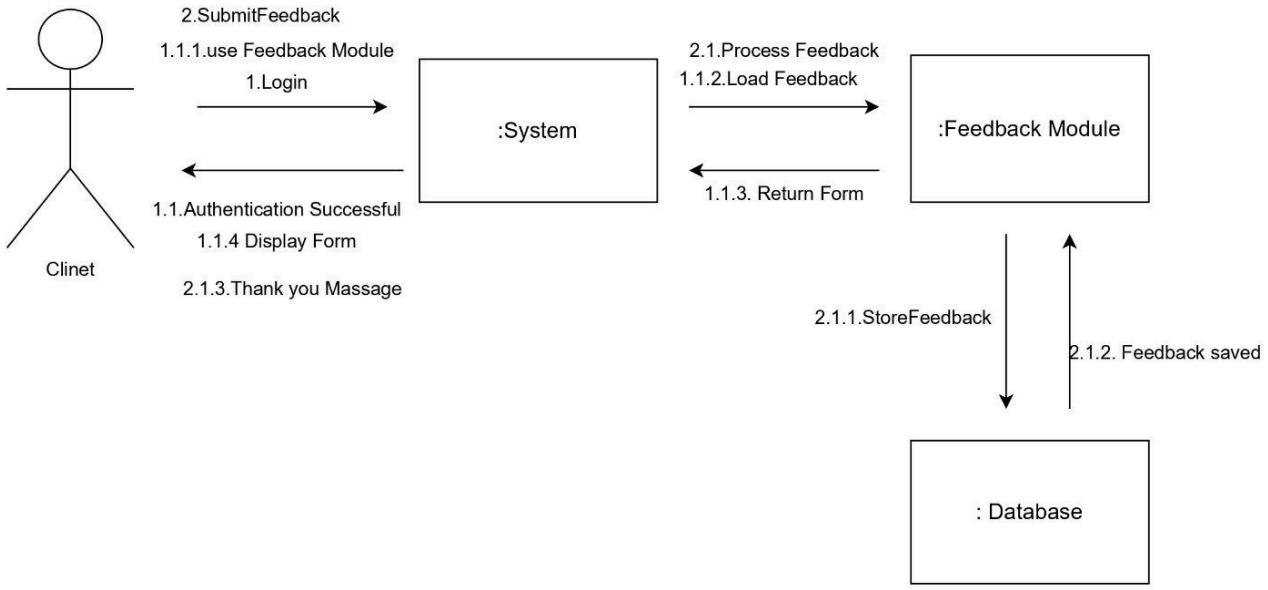
Client Requests Mechanic Assistance



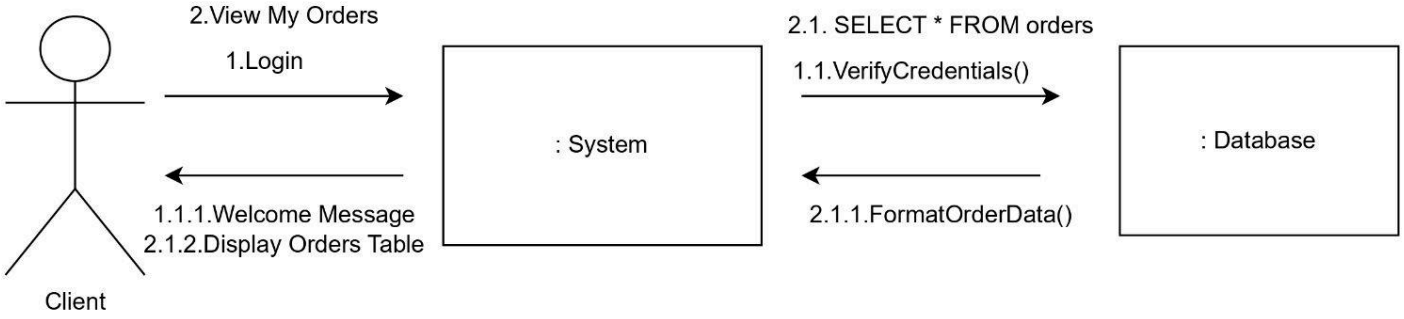
Admin Manages Mechanic Profiles



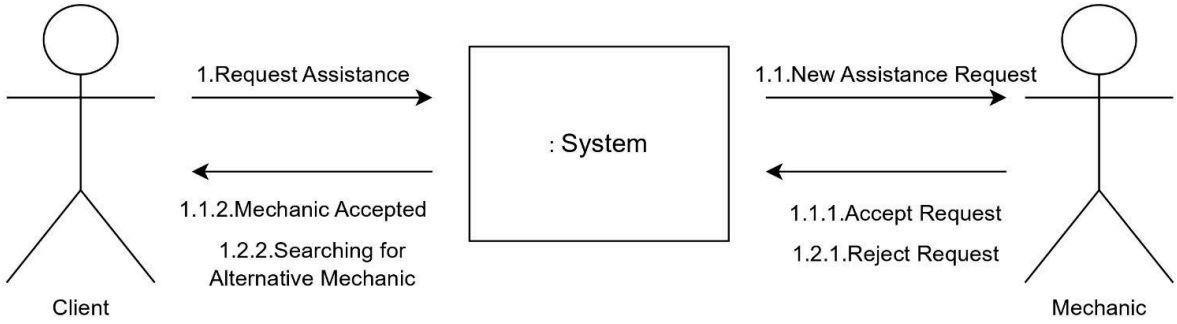
Client Provides Feedback



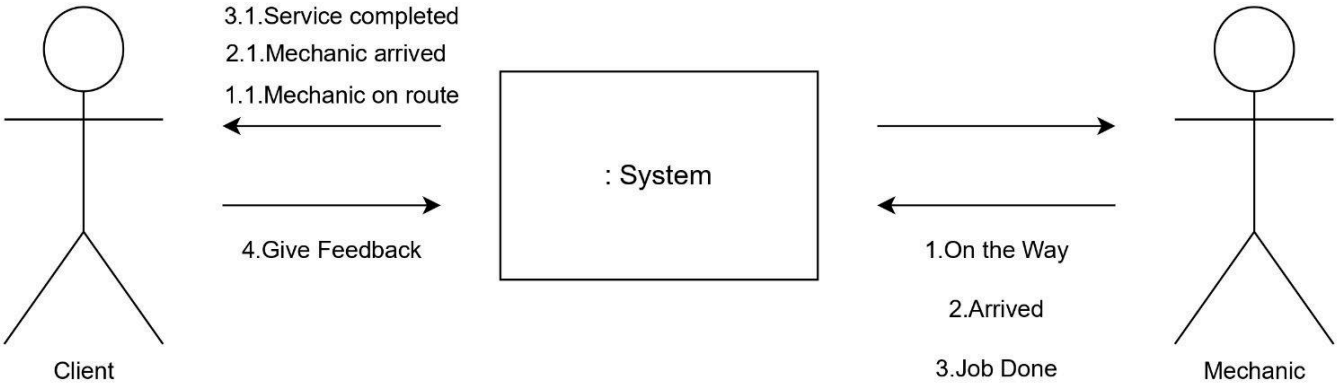
Detailed Order History View



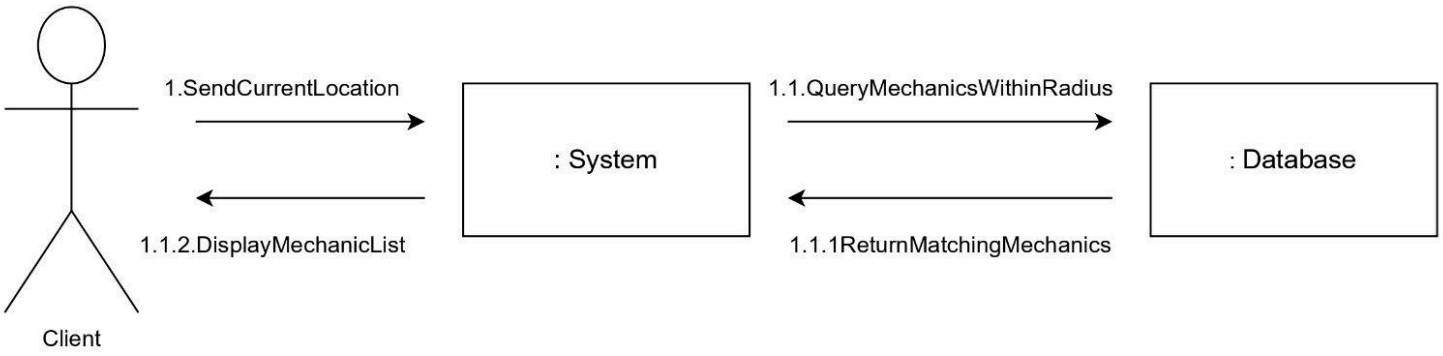
Mechanic Responds to Assistance Request



Mechanic Status Updates



System Locates Nearby Mechanics



h) Which strategy (or strategies) did you use to implement the use cases: One Central Class, Actor Class, or Use-Case class? Please explain your choice and the potential advantages/disadvantages of your design.

In implementing the system's use cases for the On-Road Vehicle Breakdown Assistance Application, we primarily adopted a combination of the Actor Class and One Central Class strategies to structure our design effectively.

1. Actor Class Strategy

We modeled each key actor in the system (Client, Mechanic, Admin) as a dedicated class that encapsulates the actions they can perform. Each class is responsible for handling the operations directly associated with the real-world role it represents—for example:

- The Client class can place and cancel orders, as well as provide feedback.
- The Mechanic class can accept assigned orders and update their status.
- The Admin class can verify mechanics and manage the system.

Advantages:

- Enhances clarity and traceability between use cases and system behavior.
- Promotes encapsulation of role-specific logic.

- Makes the system more maintainable and extensible by separating responsibilities.

Disadvantages:

- May lead to redundant logic across classes if actors share similar behaviors (e.g., login functionality).

- Can become less efficient when complex interactions require coordination across multiple actor classes.

2. One Central Class Strategy

We complemented the actor-based approach with a centralized Order class, which represents the core business process—handling the creation, assignment, status updates, and pricing of service orders. This central class acts as the coordinator of workflow, linking clients, mechanics, feedback, and status.

Advantages:

- Ensures a single source of truth for order-related logic and status transitions.

- Facilitates consistency in handling complex workflows like assigning mechanics and tracking feedback.

- Reduces duplication of process logic.

Disadvantages:

- Risk of becoming a God class if overloaded with responsibilities.

- Tightly coupling various components may affect modularity if not properly abstracted.

j) Three Design Patterns Applied (including the description for each design pattern, what problem it solves, and how it affects your system's design).

1. Singleton Pattern

Description:
The Singleton Pattern ensures that a class has only one instance and provides a global point of access to it.

Problem Solved:
It addresses the need for shared, centralized components that should only exist once in the system, avoiding redundant instances and maintaining consistent state.

Application in the System:

- The Admin class was implemented as a Singleton to ensure centralized control over system-wide operations such as managing users and mechanics.

- The LocationService was also implemented as a Singleton to manage real-time location tracking and proximity calculations from a single source of truth.

Impact on Design:

- Ensures resource consistency and centralized control.

- Reduces memory usage by avoiding redundant instances.

- Simplifies access to global services.

2. Builder Pattern

Description:
The Builder Pattern separates the construction of complex objects from their representation, allowing objects to be created step by step with a fluent and flexible interface.

Problem Solved:
It simplifies the creation of objects with many optional attributes, and improves readability and maintainability by avoiding telescoping constructors.

Application in the System:

- Mechanic and Client objects are created using dedicated builder classes (`MechanicBuilder` and `UserBuilder`). This allows for flexible and clean instantiation of these entities during the registration process or profile setup.

Impact on Design:

- Increases code clarity and flexibility during object creation.
- Improves maintainability by avoiding complex constructors.
- Makes the system easier to extend with new optional fields.

3. Observer Pattern

Description:
The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents (observers) are automatically notified and updated.

Problem Solved:
It enables automatic and decoupled communication between objects when a state change occurs, such as order status updates or notifications.

Application in the System:

- The Order class acts as an Observer to mechanic availability updates, allowing it to automatically assign a mechanic when one becomes available.

- The NotificationService uses the Observer Pattern to notify users and mechanics whenever an order status changes.

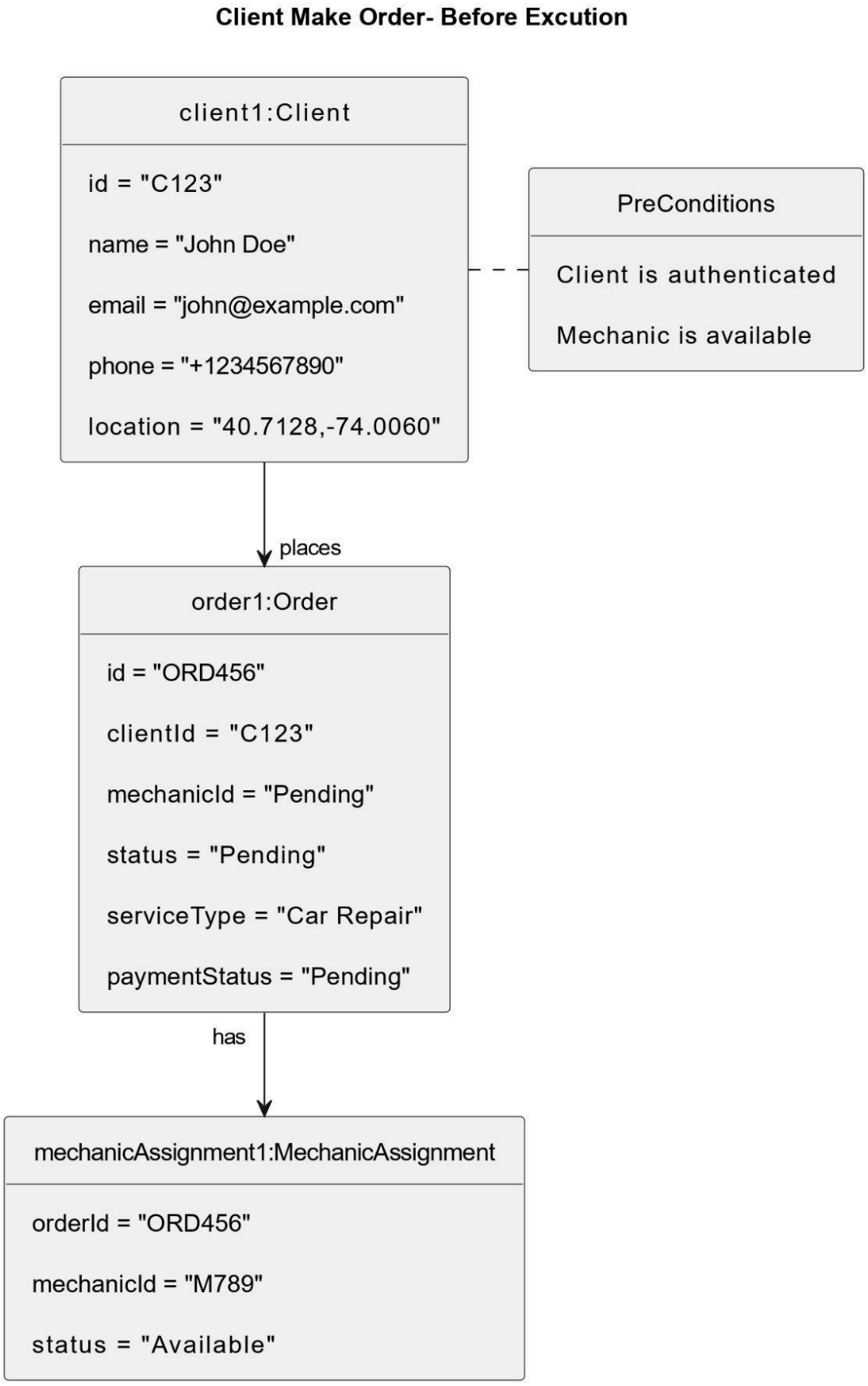
Impact on Design:

- Improves system reactivity through real-time updates.

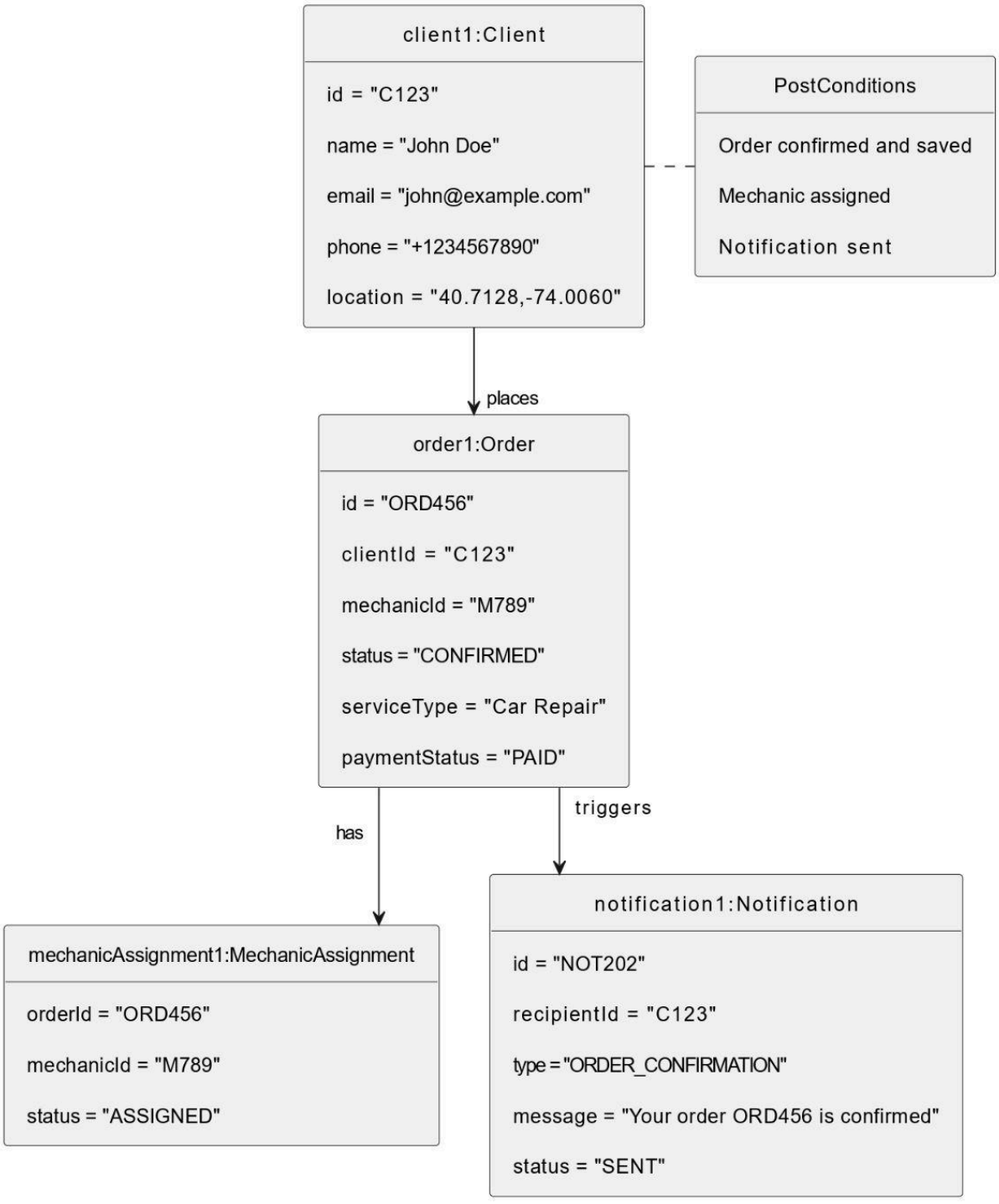
- Promotes loose coupling between components.

- Enhances user experience by providing instant notifications and automatic behavior.

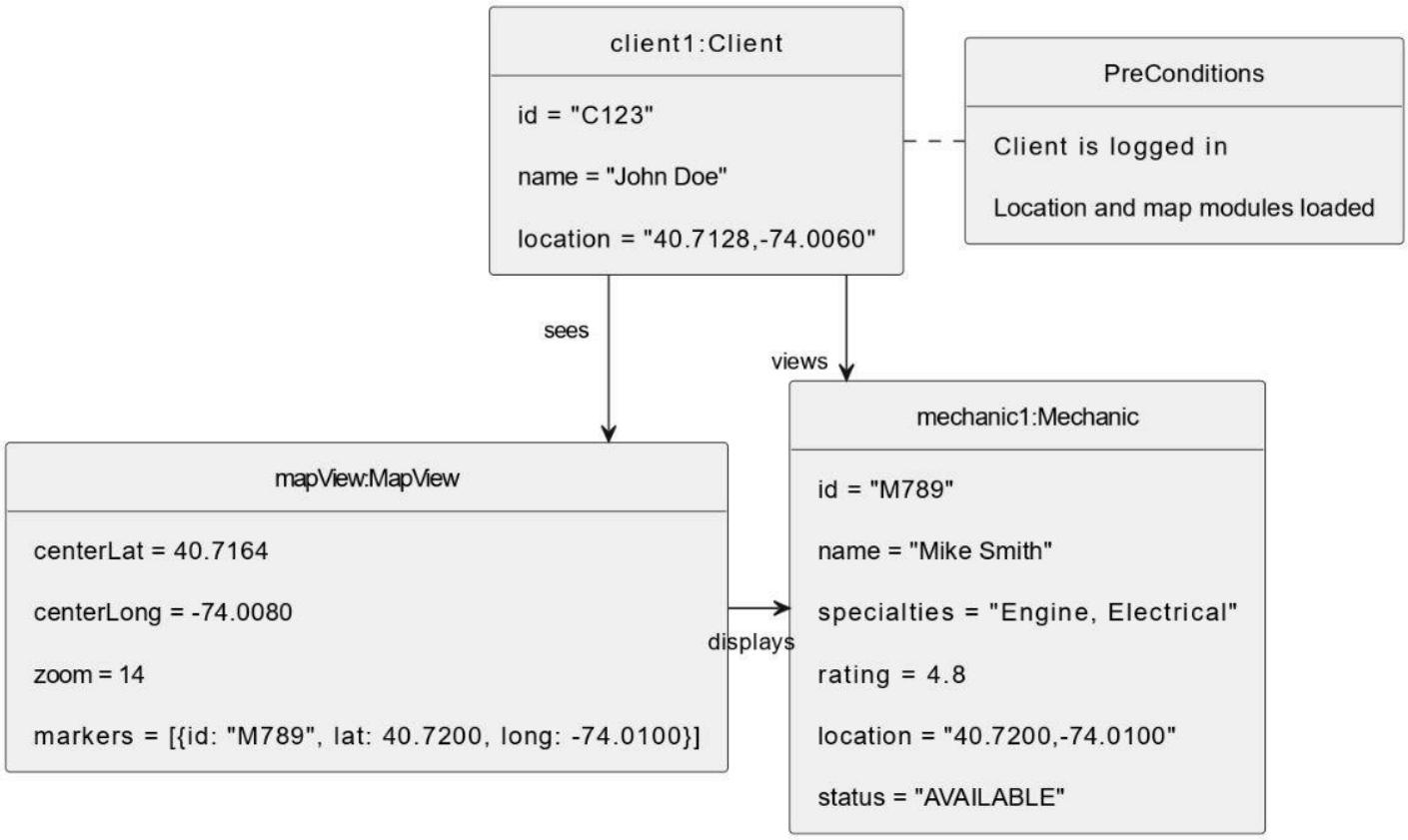
Object Diagrams



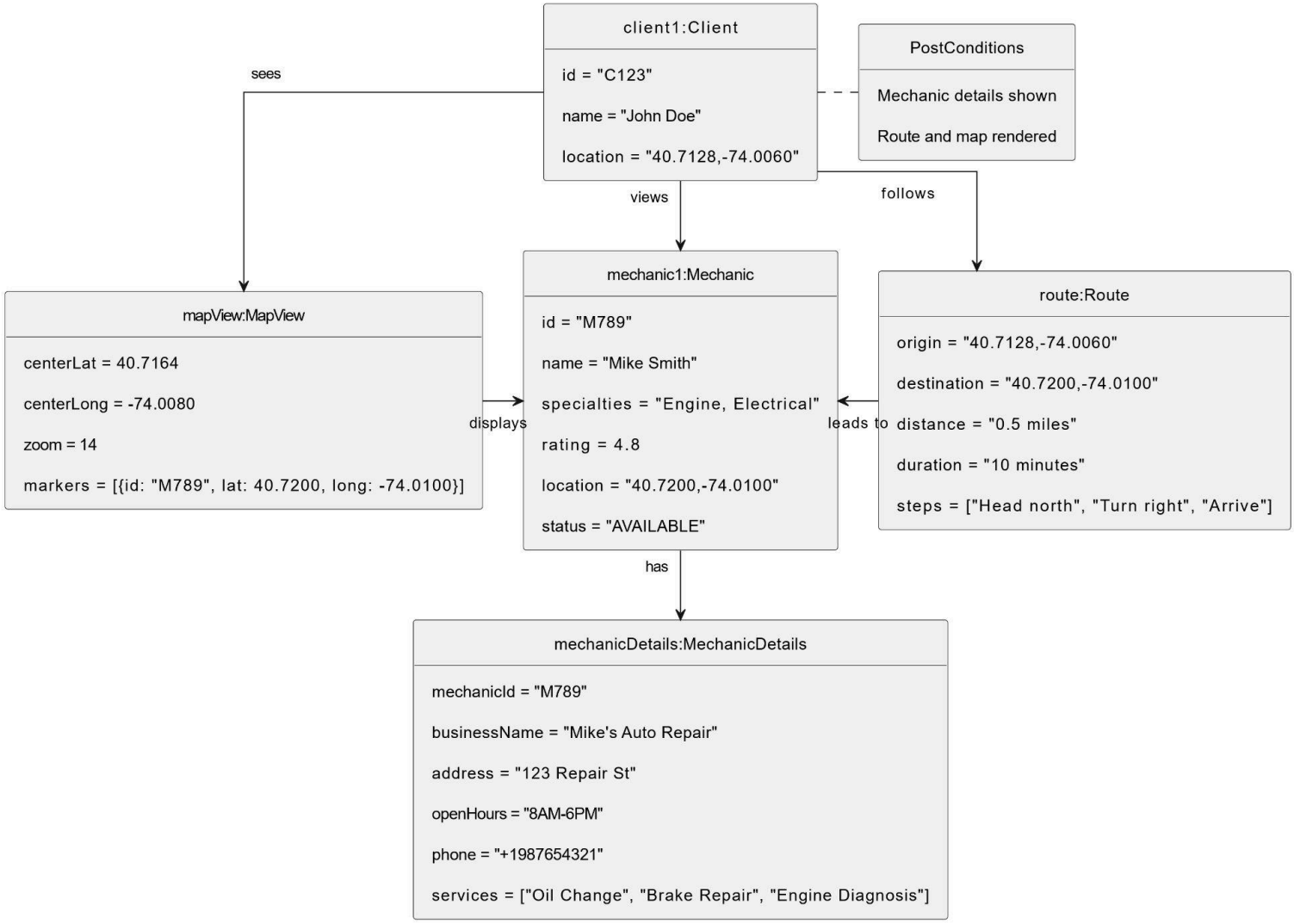
Client Make Order- After Execution



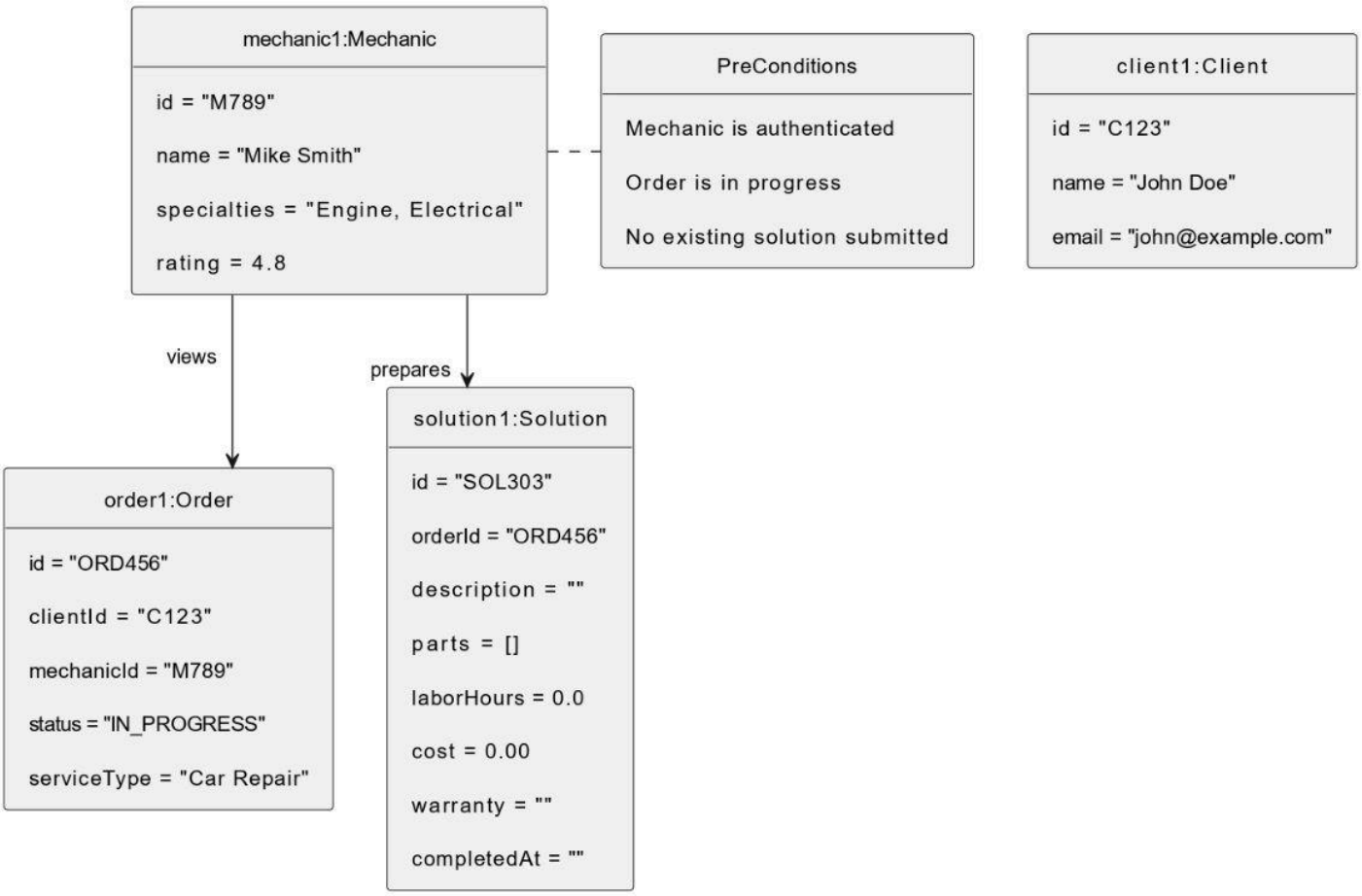
Client View Mechanic Details - Before Execution



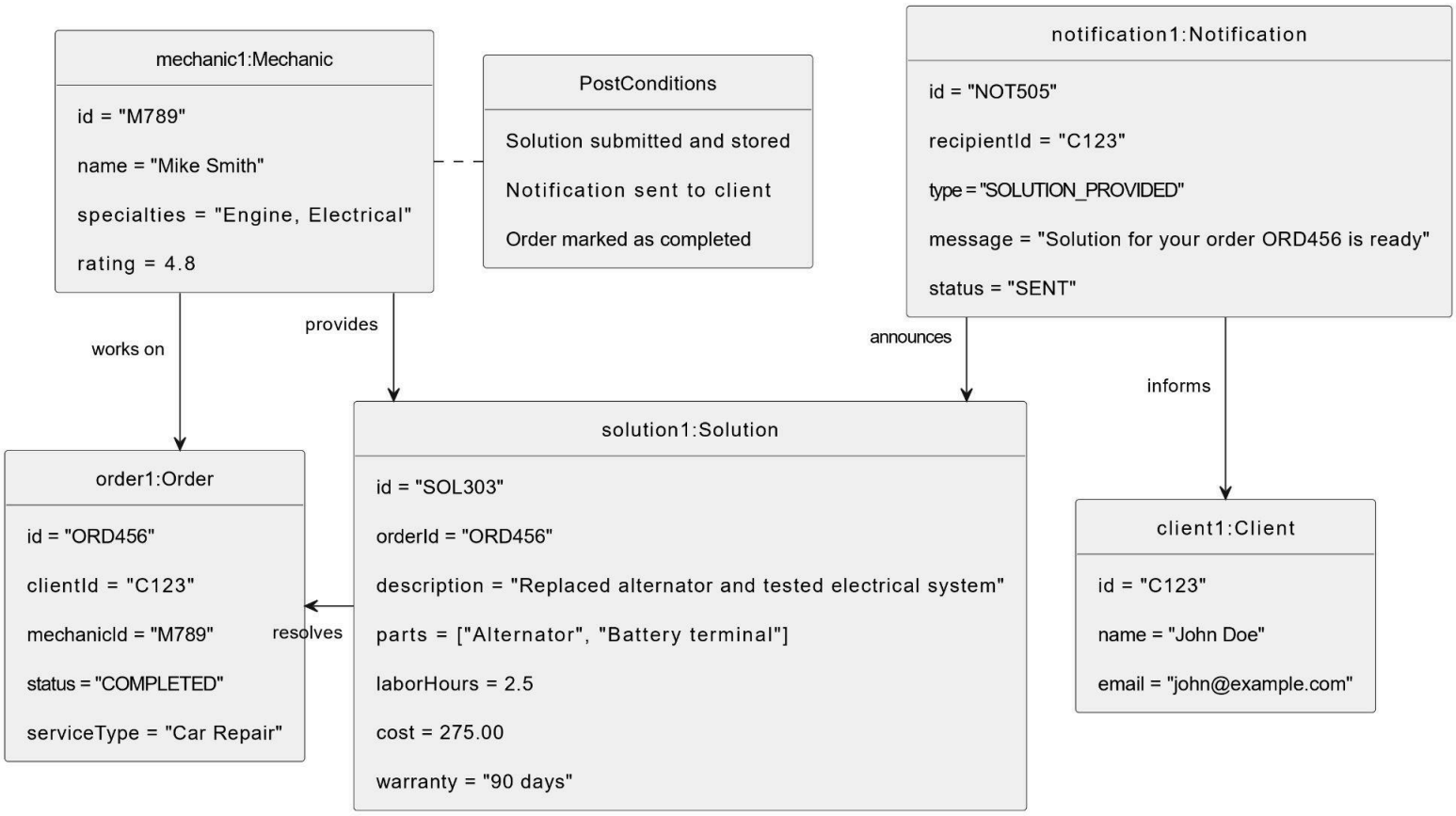
Client View Mechanic Details - After Execution



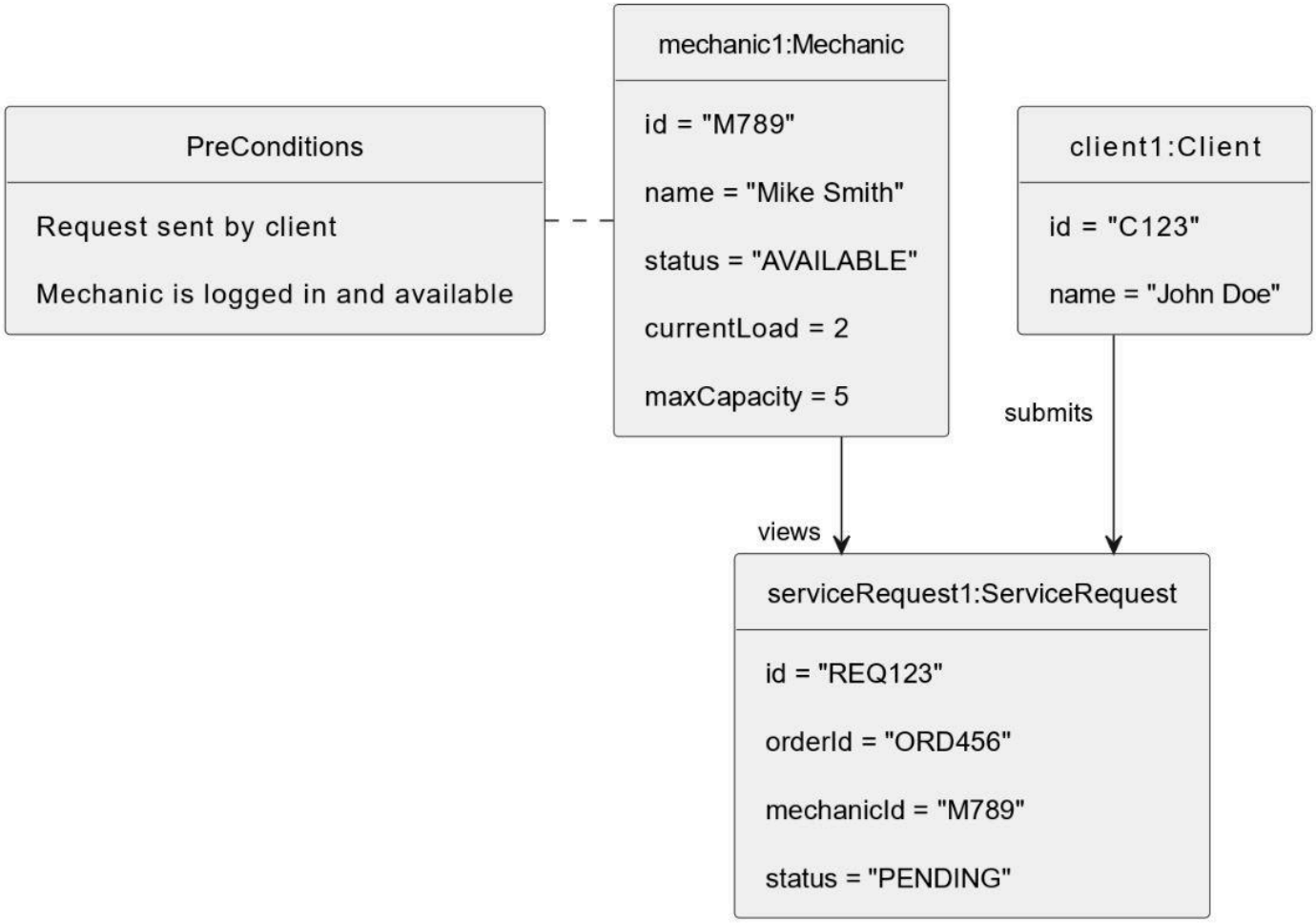
Mechanic Provide Solution - Before Execution



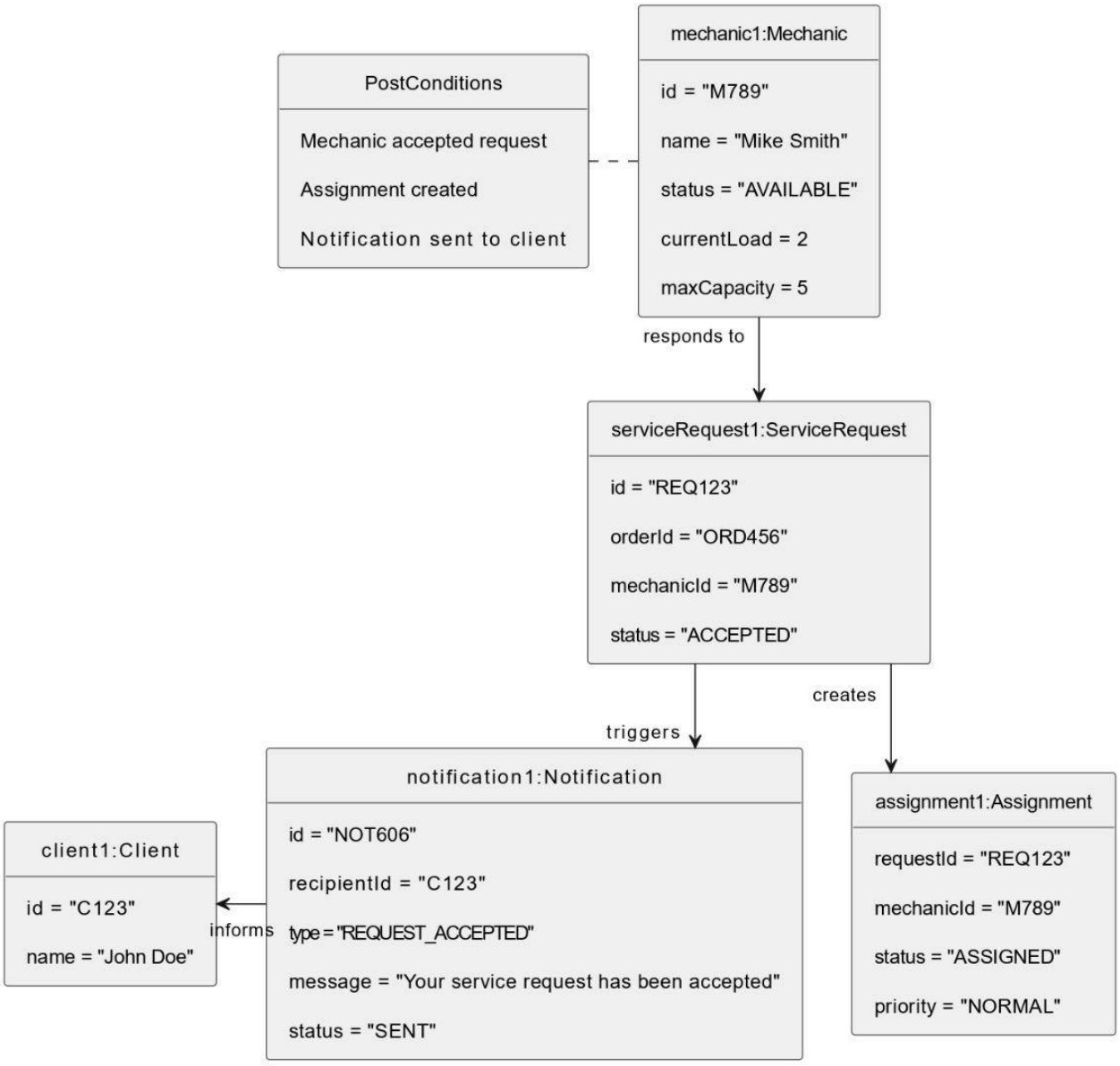
Mechanic Provide Solution - After Execution



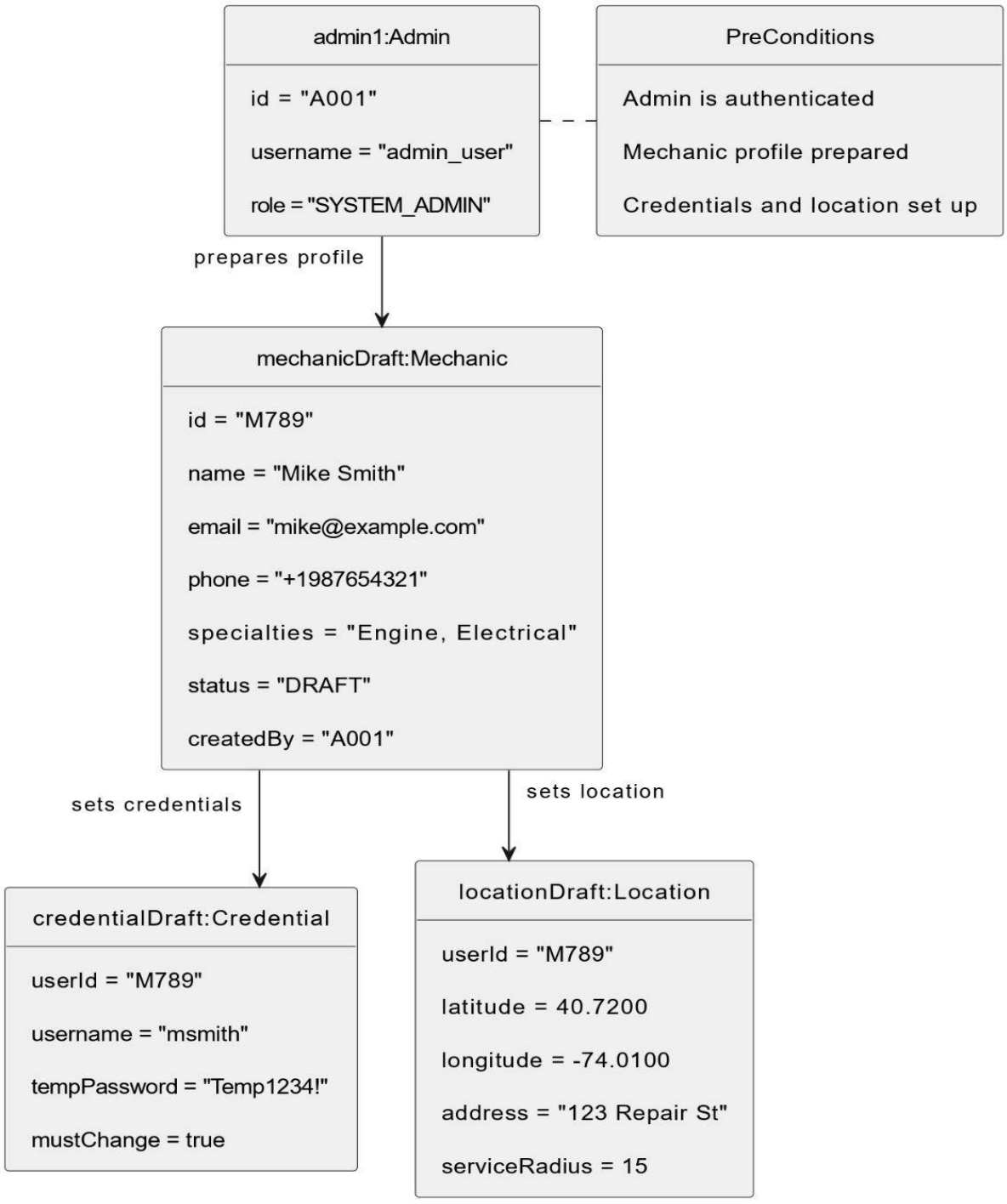
Mechanic Accept-Decline Request - Before Execution



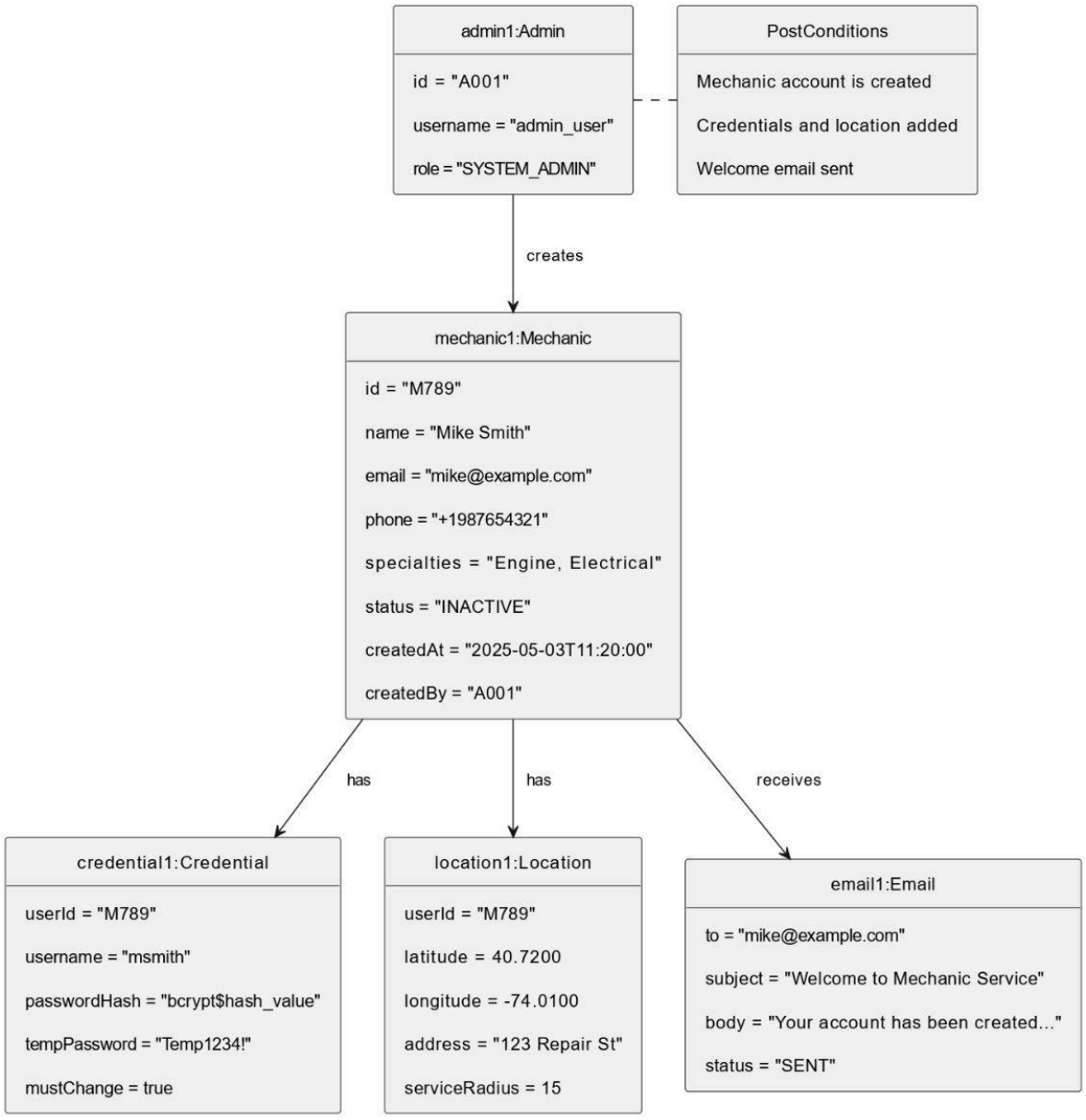
Mechanic Accept-Decline Request - After Execution



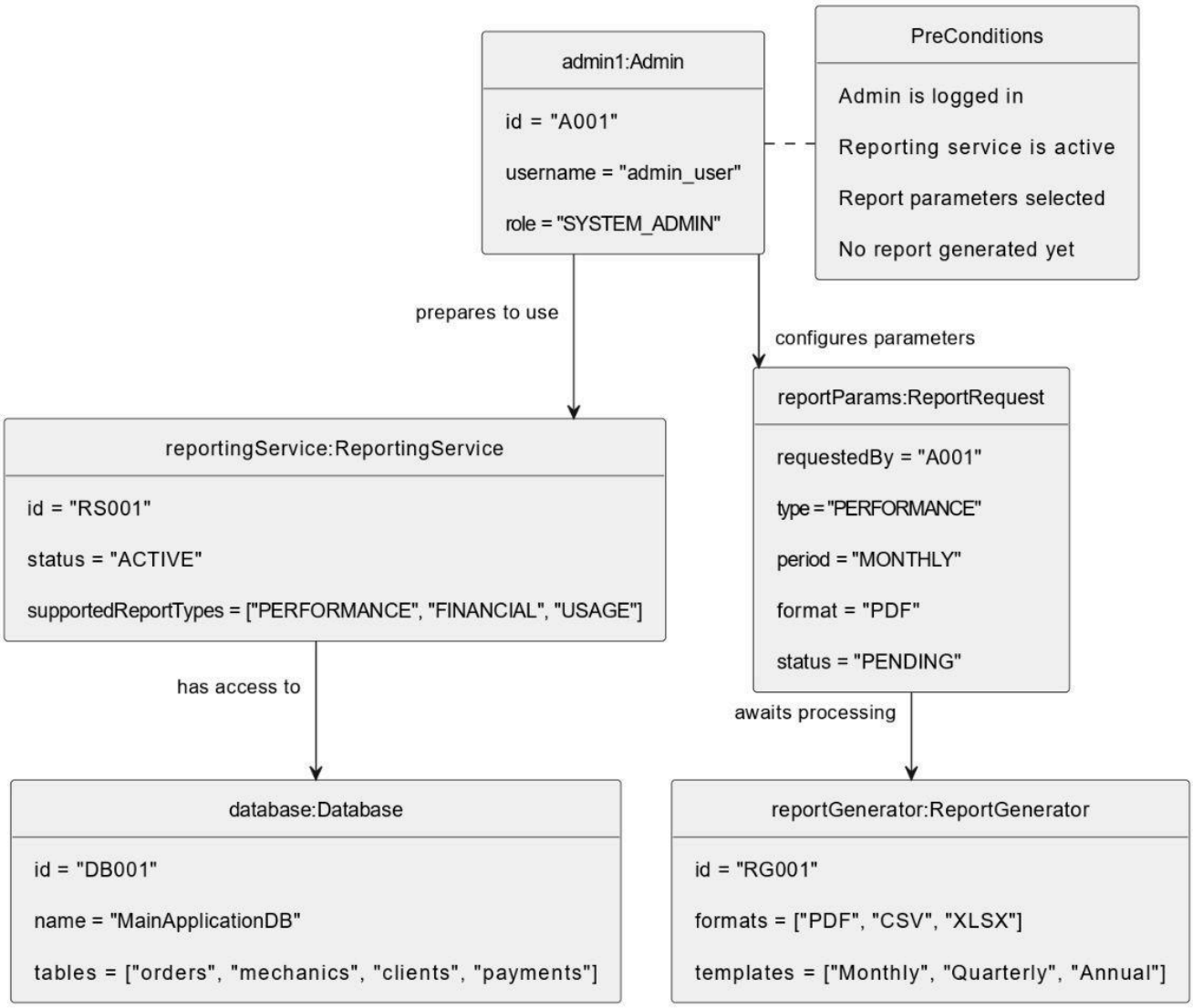
Admin Add Mechanic - Before Execution



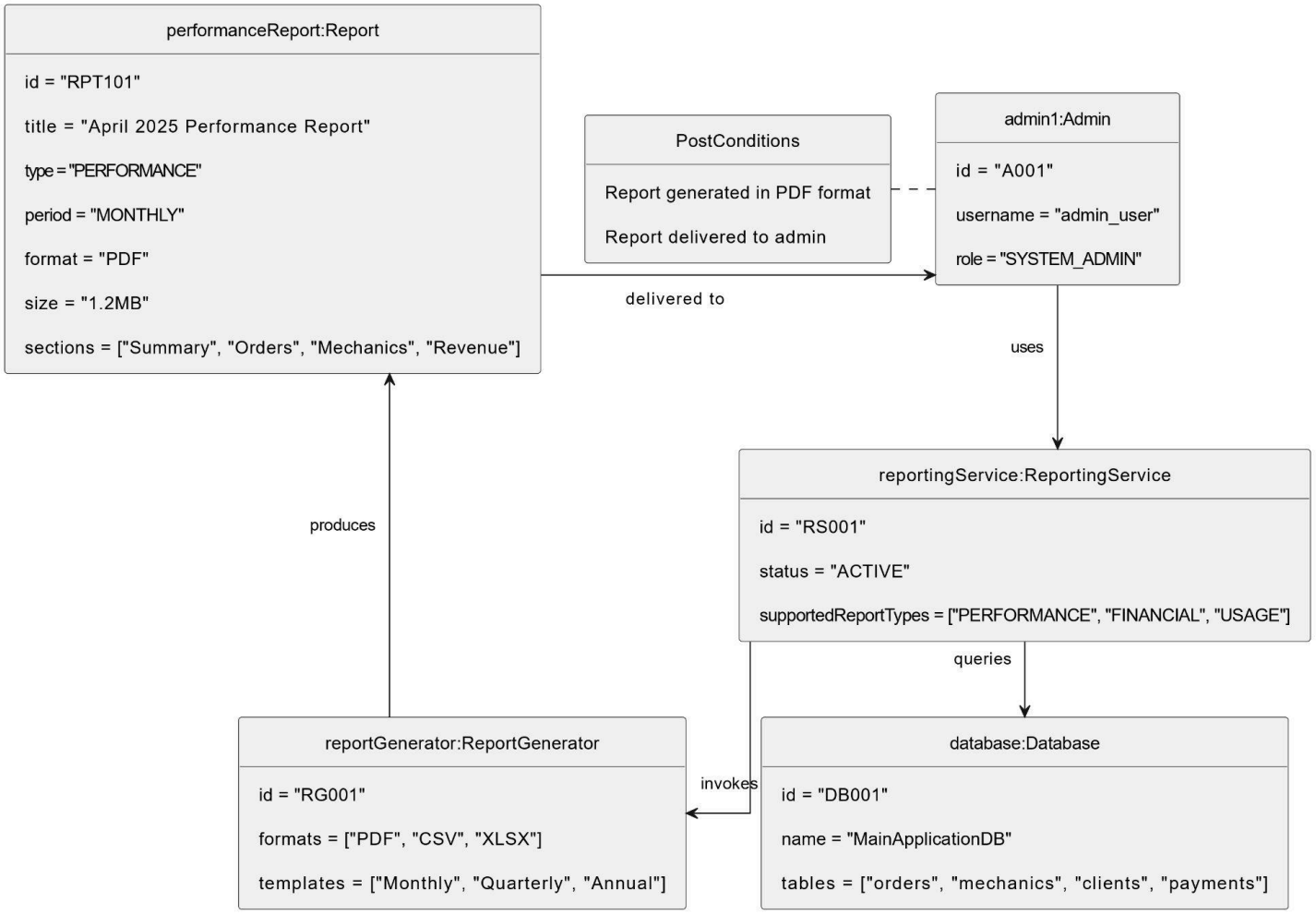
Admin Add Mechanic - After Execution



Admin Generate Performance Reports - Before Execution

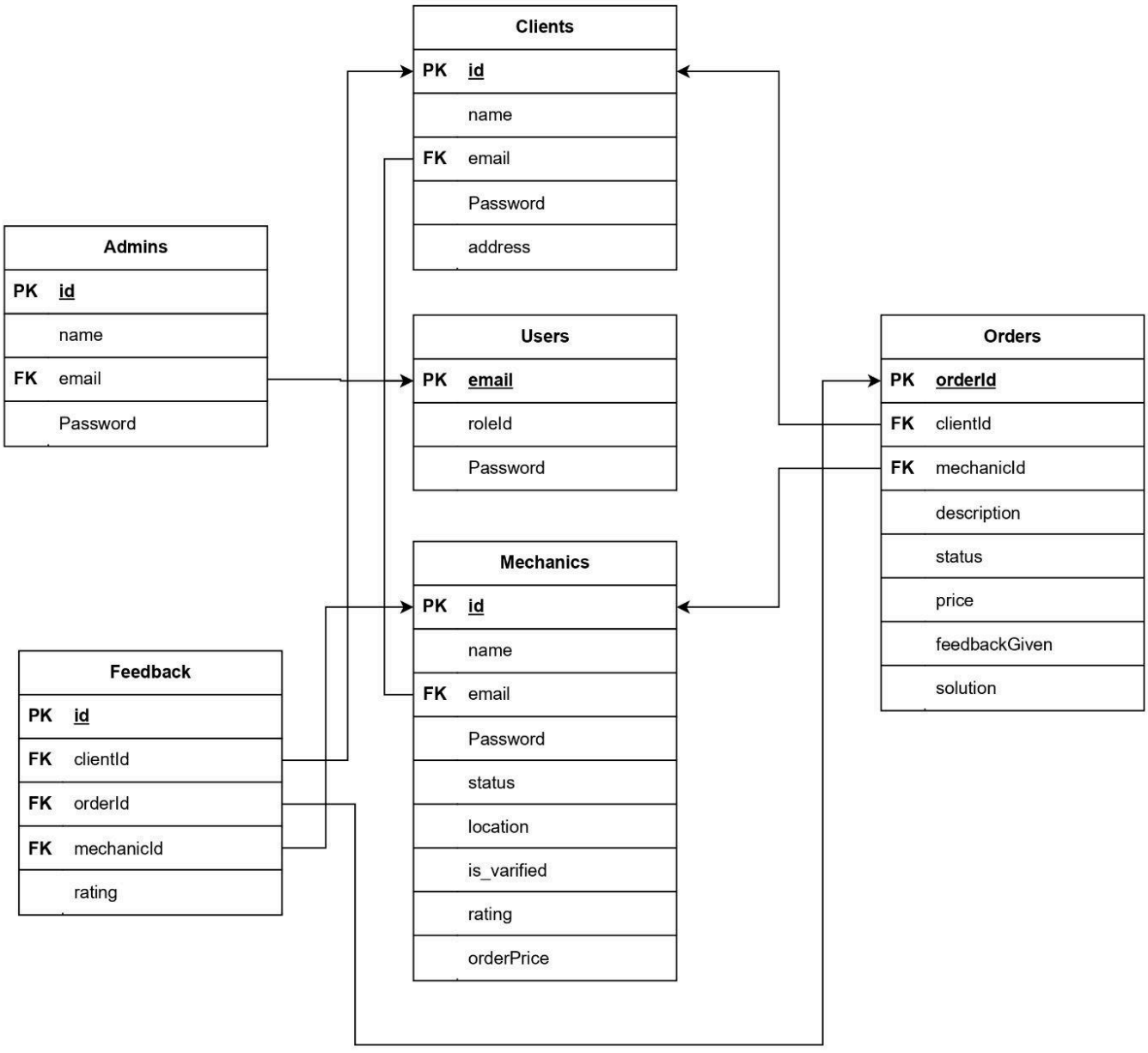


Admin Generate Performance Reports - After Execution

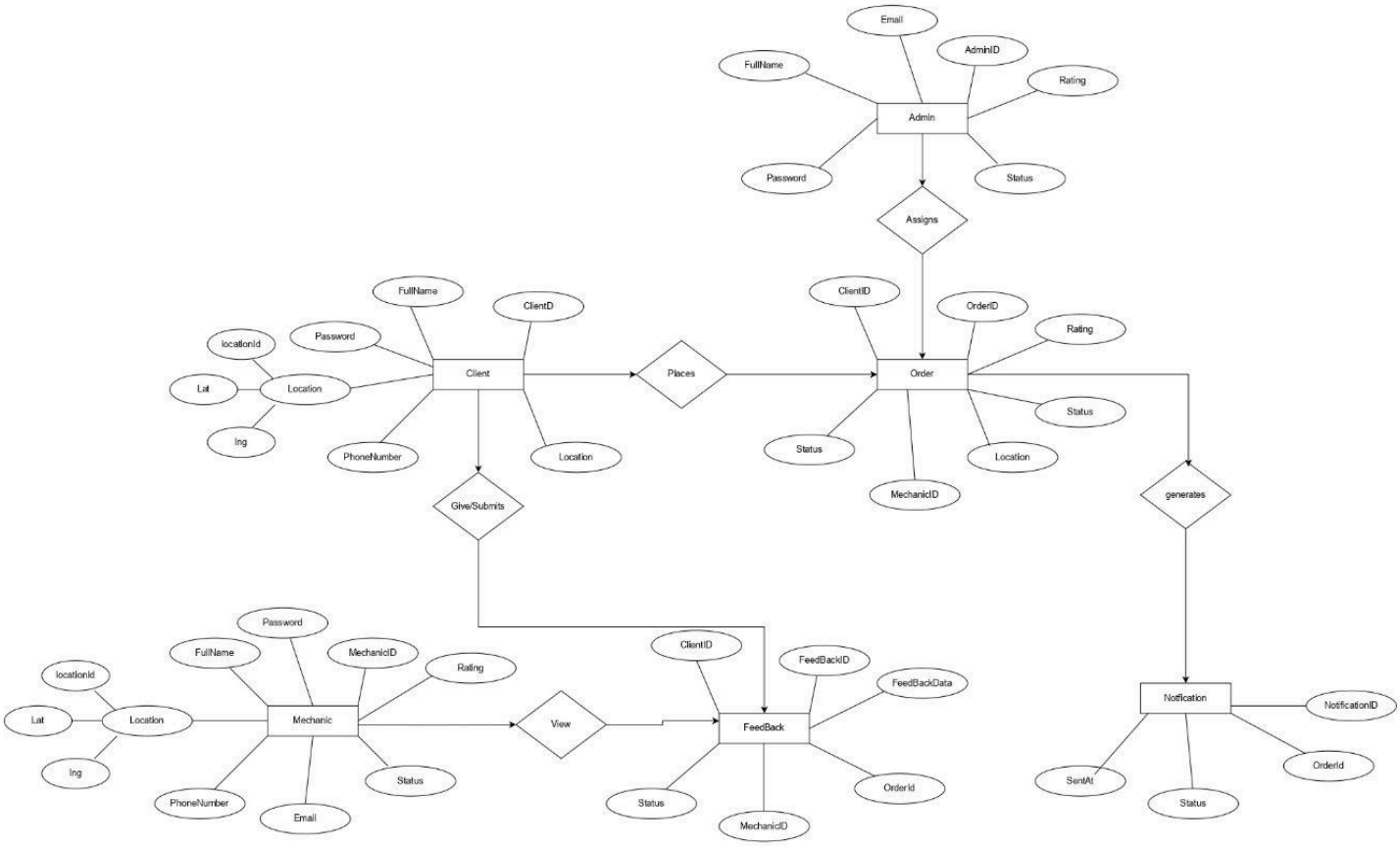


Database Specification

Database Schema



ERD


Email
AdminID
FullName
Rating
Admin
Password
Status
Assigns
FullName
ClientID
ClientID
OrderID
Rating
locationId
Password
Lat
Location
Client
Places
Order
Status
lng
PhoneNumber
Location
Status
Location
MechanicID
Give/Submits
generates
Password
FullName
MechanicID
Rating
ClientID
FeedBackID
FeedBackData
locationId
Lat
Location
Mechanic
View
FeedBack
Notfication
NotficationID
lng
Status
OrderId
SentAt
PhoneNumber
Email
Status
MechanicID
Status