Easy Park Plus

Microservices Architecture Proposal

Quantic MSSE Cohort 66

Software Design & Architecture Project

Prepared by:

Seth McKnight

Toby Pasquale

Submitted:

June 1, 2025

Executive Summary

This document defines the proposed microservices architecture for Easy Park Plus, a platform designed to **manage parking operations and integrated Electric Vehicle charging services** across urban centers.

The architecture follows Domain-Driven Design principles, **ensuring each service maps clearly to a core business function while enabling scalability, resilience, and maintainability** - supporting current operational demands and future expansion in the evolving smart urban mobility landscape.

The architecture supports Easy Park Plus' strategic goals to:

- Expand **Electric Vehicle (EV) readiness** and **idle fee enforcement** across city lots
- Improve customer experience through **real-time slot availability** and **billing transparenc**y
- Support business and fleet partnerships through **account-based access** and **validation programs**

Each service is modeled as an **autonomous bounded context with clear responsibilities, event-publishing capabilities, and independent data ownership**. All services integrate via HTTP APIs and a Kafka-based event bus. Shared infrastructure—authentication, observability, fallback behavior, and platform-wide policies—is handled by lightweight supporting services.

Key domains include:

- Parking Operations
- Electric Vehicle Charging
- Reservations
- Customer, Vehicle, and Policy Management
- Billing & Pricing

A Minimum Viable Product (MVP) rollout will focus on launching six core services—Parking Operations, Electric Vehicle Charging, Billing, Customer, Vehicle, and Policy Management—enabling full end-to-end parking and EV workflows in a pilot lot.

This proposal defines a **robust, agile, forward-looking foundation** for future enhancements, including curbside pickup, bicycle infrastructure, and vehicle-to-grid support.

1) Introduction & Architectural Vision

This document defines the architecture strategy for Easy Park Plus—a distributed platform designed to operate multi-story parking facilities and Electric Vehicle charging infrastructure across Boston, New York, Philadelphia, and future cities.

Domain-Driven Design principles divide the platform into cohesive, autonomous services.

Each service represents a bounded context tied to a specific business capability—such as Parking Operations, Billing, or EV Charging—and aligns to the way Easy Park Plus teams and workflows are structured.

1.1) Architectural Vision

The vision is to create a system that is:

- **Scalable**: Services operate and scale independently across regions or lots
- **Resilient**: Localized failures do not cascade
- **Extensible**: Capable of supporting new capabilities like bicycle storage, curbside, etc.
- **Policy-Driven**: Behavior and pricing governed by rule sets without redeploying code
- **Observable**: All actions logged, metrics are emitted, & decision paths are auditable
- **Customer-Centric**: Slot availability, reservations, & validations are first-class features

The architecture empowers Easy Park Plus to:

- **Expand rapidly** into new cities and lots
- **Integrate new services** such as Electric Vehicle idle fee enforcement
- **Support high-availability** infrastructure like ANPR-based vehicle access
- **Enable differentiated business models** through:
    - Flexible pricing
    - Account-based access
    - Service-level policies

This vision provides a foundation for operational excellence today and adaptability to the urban mobility challenges of tomorrow.

2) Bounded Contexts Overview

This section provides a concise overview of the Bounded Contexts that reflect the business and operational structure of Easy Park Plus. “Each bounded context corresponds to a cohesive business capability—such as Parking Operations, Billing, or EV Charging—and is aligned to operational units or functional teams at Easy Park Plus.” They are expanded on with Ubiquitous Language, Key Aggregates, and APIs/Events in Section 3 (Service Designs).

- **Parking Operations Context:** Manages the real-time vehicle visit lifecycle, slot status, and operational policies.
- **Electric Vehicle Charging Context:** Manages all aspects of EV charging sessions and hardware.
- **Reservations Context:** Manages pre-booking of parking spots and chargers, including temporary slot holds.
- **Customer Context:** Manages customer identities, profiles, vehicles, and passholder status.
- **Vehicle Context:** Acts as the central registry for vehicle information and status.
- **Pricing Context:** Determines how much to charge based on rules, conditions, and policies. Outputs charge estimates and previews.
- **Billing Context:** Handles when, how, and by whom charges are settled, including payment processing, invoices, and refunds.
- **Infrastructure Management Context:** Owns all static physical structure and relationships between lots, levels, slots, gates, and fixed equipment.
- **Policy Management Context:** Stores and applies operational policies across Parking, EV, Validation, and Valet services, including customer-type specific rules.
- **Staff & User Access Context:** Manages employee and partner roles, permissions, and audit logging of staff actions.
- **System Configuration Context:** Manages internal system-level configurations, operational modes (like Fallback Policies), and overall system health parameters not tied to specific external APIs.
- **External Integrations Context:** Manages the configuration, pre-defined contracts, connectivity, and status monitoring of all external third-party hardware and software APIs.
- **Incident & Enforcement Context:** Manages rule violations, incidents, equipment faults, and enforcement workflows.
- **Analytics & Reporting Context:** Consumes data for historical analysis, forecasting (including slot availability), and reporting.
- **Internal Audit & Event Trace Context:** Subscribes to all domain events for comprehensive logging, compliance, debugging, and SLA enforcement.

3) Service Designs - Bounded Contexts and Microservices

This section provides summaries for each core Bounded Context and its corresponding Microservice. It details the individual microservices that form the Easy Park Plus platform, outlining the specific domain and responsibilities of each.

Each subsection provides details on the service's purpose, ubiquitous language, key aggregates, conceptual core APIs, main published events, related policies consumed, and important notes or dependencies.

3.1) Parking Operations

- **Purpose:** Manages the active lifecycle of vehicles inside a parking facility. Responsible for visit creation, gate access, slot assignment (including various types like OpenAccess, Rented, Valet), real-time occupancy tracking, fallback handling during system degradations, conflict detection, and managing the VisitLifecycleStateMachine. It also logs valet activities and handles reconciliation of visits logged in degraded modes.
- **Ubiquitous Language:** Visit, Entry, Exit, Slot, Occupancy, Status, Gate, Fallback, Reconciliation, Manual Override, Abandoned Vehicle, Emergency Mode, ValetVisit, ValetActivity, VisitState, GateFault.
- **Key Aggregates:** Visit, Gate (Operational View), Slot (Operational View), ValetActivity, VisitReconciliationJob.
- **Core APIs (Conceptual):**
  - POST /visits: Create a new Visit on entry.
  - PUT /visits/{visitId}/slot: Assign/update slot for a Visit.
  - POST /visits/{visitId}/exit: Record vehicle exit, complete Visit.
  - GET /lots/{lotId}/slots/status: Get real-time status of slots in a lot.
  - POST /valet/activities: Log a new valet activity.
- **Main Events (Published):**
  - VehicleEnteredLot (Vehicle enters via ANPR or manual override)
  - VehicleExitedLot (Vehicle exits, signals billing/finalization)
  - SlotAssignedToVisit (Vehicle is assigned to a specific slot)
  - VisitReadyForPricing (Visit details complete for price quote)
  - GateFaultDetected (Gate hardware fault)
  - VisitLoggedUnderFallback (Visit processed during system degradation)
- **Related Policies (Consumed):** GracePeriodPolicy (entry grace), FallbackPolicy, ValetServicePolicy, SlotAssignmentType rules.
- **Notes:** Handles fallback, reconciliation, conflict detection. Integrates with Reservations, Pricing, Billing, Infrastructure, Policy, System Config, External Integrations, Analytics, Incident & Enforcement. Valet staffing may use OccupancyForecast.

3.2) Electric Vehicle Charging

- **Purpose:** Manages all aspects of Electric Vehicle charging sessions, including charger hardware status (via External Integrations), energy consumption metering, application of Electric Vehicle specific rules like idle fees (based on GracePeriodPolicy), and handling of charging interruptions or faults.
- **Ubiquitous Language:** Charger, Charging Session, KilowattHour, Idle Time, Grace Period, Fault, Connector Type, ElectricVehicle, Interruption, ChargerFault.
- **Key Aggregates:** ChargingSession, Charger (Operational View).
- **Core APIs (Conceptual):**
  - POST /sessions: Start a new charging session.
  - PUT /sessions/{sessionId}/energy: Record energy delivery updates.
  - DELETE /sessions/{sessionId}: End/unplug a charging session.
  - GET /chargers/{chargerId}/status: Get real-time charger status.
- **Main Events (Published):**
  - ChargingSessionStarted (EV plugged in and session initiated)
  - ChargingSessionEndedAndReadyForBilling (Session concluded)
  - ChargerFaultDetected (Charger hardware fault)
  - IdleTimeExceeded (Grace period ended post-charging)
- **Related Policies (Consumed):** GracePeriodPolicy (EV idle times), EVChargingPolicy, FaultBillingAdjustmentPolicy.
- **Notes:** Relies on External Integrations for charger hardware APIs. Charger status influenced by Infrastructure Management and Incident & Enforcement.

3.3) Reservations

- **Purpose:** Manages pre-booking of parking spots and EV chargers for short-term use (DynamicReservation), including temporary SlotHold requests. Ensures availability by filtering out long-term assigned slots and considering OccupancyForecast.
- **Ubiquitous Language:** DynamicReservation, Electric Vehicle Reservation, Booking, Hold, SlotHold, Expiry, Confirmation, Slot Type, Time Window, Mismatch, Forecast, ReservationGroup.
- **Key Aggregates:** DynamicReservation, SlotHold.
- **Core APIs (Conceptual):**
  - POST /reservations/holds: Create a temporary slot hold.
  - POST /reservations: Create/confirm a dynamic reservation.
  - GET /reservations/{reservationId}: Retrieve reservation details.
  - DELETE /reservations/{reservationId}: Cancel a reservation.
- **Main Events (Published):**
  - SlotHoldCreated (Request to temporarily reserve a slot)
  - SlotHoldExpired/Released (Hold time limit reached or released)
  - DynamicReservationCreated (New reservation booked)
  - DynamicReservationFulfilled (Reserved vehicle arrived)

- **Related Policies (Consumed):** EventWindowPolicy, CustomerTypeSpecificRulePolicy (reservation limits).
- **Notes:** SlotHold improves UX. Uses OccupancyForecast from Analytics.

3.4) Customer

- **Purpose:** Manages customer identities, profiles (Individual, Business, Fleet, etc.), linked vehicles, tokenized payment methods, passholder status with access rules, and commuter benefits. Links sponsoring businesses to validation programs.
- **Ubiquitous Language:** Customer, Profile, Vehicle Link, Passholder, Membership, Payment Method, Commuter Benefit, Access Rule, CustomerType, ValidationProgramLink, SponsoringBusiness, ValetProvider.
- **Key Aggregates:** Customer.
- **Core APIs (Conceptual):**
    - POST /customers: Register a new customer.
    - GET /customers/{customerId}: Retrieve customer profile.
    - POST /customers/{customerId}/vehicles: Link a vehicle.
    - POST /customers/{customerId}/passholder: Enroll in a passholder program.
- **Main Events (Published):**
    - CustomerRegistered (New customer account created)
    - VehicleLinkedToCustomer (Vehicle associated with account)
    - PaymentMethodTokenAddedToCustomer (New payment token linked)
    - PassholderStatusUpdated (Pass status or access rules change)
- **Related Policies (Consumed/Referenced):** ValidationProgram configurations.
- **Notes:** Payment details are tokenized (actuals in Billing). CustomerType is key for type-specific rules.

3.5) Vehicle

- **Purpose:** Central registry for all vehicle information: characteristics, EV capabilities, and system-wide status (Active, Banned, etc.). Status updated via events. Includes logic for potential plate cloning detection.
- **Ubiquitous Language:** Vehicle, Registration Plate, Manufacturer, Model, Type, Electric Vehicle Capability, Status, Ban, Plate Cloning.
- **Key Aggregates:** Vehicle.
- **Core APIs (Conceptual):**
    - POST /vehicles: Register a new vehicle.
    - GET /vehicles/{vehicleRegistration}: Retrieve vehicle details and status.
    - PUT /vehicles/{vehicleRegistration}/details: Update vehicle characteristics.
- **Main Events (Published):**
    - VehicleRegisteredInSystem (Vehicle's first interaction)
    - VehicleSystemStatusUpdated (Vehicle status changes, e.g., banned)
    - PlateCloningSuspected (Anomalous activity detected)

- **Related Policies:** Vehicle data informs policy application in other contexts.
- **Notes:** Crucial for ANPR, slot assignment, Electric Vehicle charging, enforcement.

3.6) Pricing

- **Purpose:** Determines charges for parking, EV charging, etc. Owns pricing rules, rate definitions. Handles adaptive pricing. Provides charge estimates/previews. Consumes pricing-related policies.
- **Ubiquitous Language:** Price, Rate, Fee, Discount, TaxRule, PricingRuleSet, RateDefinition, ChargeEstimate, PricingPreview, AdaptiveTrigger.
- **Key Aggregates:** PricingRuleSet.
- **Core APIs (Conceptual):**
    - POST /pricing/estimate/visit: Calculate estimated cost for a visit.
    - POST /pricing/estimate/ev-session: Calculate estimated cost for an EV session.
    - GET /pricing-rules/active: Retrieve active pricing rules.
- **Main Events (Published):**
    - PriceQuoteCalculatedForVisit (Signals Billing with charge breakdown)
    - PriceQuoteCalculatedForEVSession (Signals Billing with charge breakdown)
- **Related Policies (Consumed):** EventWindowPolicy, ValidationProgramPolicy, CustomerTypeSpecificPricingPolicy.
- **Notes:** Stateless evaluation engine. Consumes OccupancyForecast for dynamic pricing.

3.7) Billing

- **Purpose:** Handles financial settlement. Owns Bills, tokenized PaymentMethods, processes Transactions, issues Invoices, handles refunds. Applies ParkingValidation redemptions.
- **Ubiquitous Language:** Bill, Payment, Transaction, Invoice, Refund, Chargeback, Settlement, PaymentMethod, PassholderBillingCycle, RentedSlotFee, ValidationRedemption.
- **Key Aggregates:** Bill, PaymentMethod, Transaction, Invoice, ValidationRedemption, PassholderBillingCycle, RentedSlotFeeLedger.
- **Core APIs (Conceptual):**
    - POST /bills: Create a bill.
    - POST /bills/{billId}/payments: Attempt payment.
    - POST /validations/redeem: Record/apply a validation.
- **Main Events (Published):**
    - BillCreated/BillIssued (Bill finalized/issued)
    - PaymentSucceeded (Payment gateway confirms success)
    - PaymentFailed (Payment attempt declined/failed)
    - ValidationAppliedToBill (Validation successfully applied)
- **Related Policies (Consumed):** GracePeriodPolicy (renewals), FleetInvoiceAggregationPolicy, FaultBillingAdjustmentPolicy.

- **Notes:** PCI DSS scope concentrated here. Integrates with payment gateways via External Integrations.

3.8) Infrastructure Management

- **Purpose:** Owns static physical definition of parking infrastructure: Lots, Levels, SlotConfiguration, Gates, fixed EquipmentInventoryItems. Source of truth for "what is where".
- **Ubiquitous Language:** Lot, Level, Slot, Gate, Equipment, PhysicalLayout, Configuration, Asset.
- **Key Aggregates:** Lot, Level, SlotConfiguration, Gate, EquipmentInventoryItem.
- **Core APIs (Conceptual):**
    - GET /lots/{lotId}: Retrieve lot structure.
    - GET /slots/{slotId}: Get slot configuration.
    - Admin APIs for POST/PUT to manage these entities.
- **Main Events (Published):**
    - LotCreated/Updated (Lot details changed)
    - SlotConfigurationChanged (Slot attributes modified)
    - EquipmentItemStatusChanged (Maintenance status of equipment updated)
- **Related Policies:** Defines physical structure upon which policies are scoped.
- **Notes:** Primarily read-heavy. Writes require strict control. Caching is vital.

3.9) Policy Management

- **Purpose:** Defines, manages, and evaluates operational policies (Grace Periods, Event Windows, Validation, Valet, EV Charging, Customer-Type Specific Rules, Fault Billing Adjustments). Handles policy scoping and override hierarchies. Provides effective policies.
- **Ubiquitous Language:** Policy, RuleSet, Scope, Override, GracePeriod, EventWindow, ValidationPolicy, ValetPolicy, EVPolicy, CustomerTypePolicy, FleetBillingPolicy, FaultAdjustmentPolicy.
- **Key Aggregates:** GracePeriodPolicy, EventWindow, ValidationProgramPolicy, ValetServicePolicy, CustomerTypeSpecificRulePolicy, FaultBillingAdjustmentPolicy, ValidationProgram, ValetProvider.
- **Core APIs (Conceptual):**
    - GET /policies/effective?context_params...: Centralized endpoint to resolve effective policies.
    - Admin APIs (POST/PUT /policies/{policyType}, etc.) for managing policies.
- **Main Events (Published):**
    - PolicyUpdated (or specific policy type) (Any policy configuration changes)
    - ValidationProgramUpdated (Validation program details change)
    - ValetProviderUpdated (Valet provider operational parameters change)
- **Related Policies:** This context *is* the source of truth for most operational policies.

- **Notes:** Policy resolution is centralized and critical. High performance and caching are essential.

3.10) Staff & User Access

- **Purpose:** Manages identity, authentication, and authorization for all user types. Owns UserAccounts, Roles, PermissionSets, and a comprehensive UserActivityLog.
- **Ubiquitous Language:** UserAccount, UserType, Customer, Staff, Partner, System, UserIdentity, Role, Permission, PermissionSet, AccessControl, Authentication, Authorization, UserActivityLog.
- **Key Aggregates:** UserAccount (typed: Customer, Staff, Partner, System), Role, UserActivityLog.
- **Core APIs (Conceptual):**
  - POST /auth/login: Authenticate any user
  - POST /auth/customer/register: Register a new customer account
  - GET /users/me: Get current user's access-relevant profile and permissions
  - POST /admin/users: Create staff/partner/system user
  - POST /admin/roles: Create role
  - PUT /admin/roles/{roleId}/permissions: Assign permissions to role
- **Main Events (Published):**
  - UserAccountCreated (with user type)
  - UserAccountUpdated (e.g., role/status change, password reset initiated)
  - UserLoggedIn (with user type)
  - UserRegistered (customer self-registration)
  - UserActionLogged (actions by any authenticated user, filterable by type)
- **Related Policies (Consumed/Enforced):** Enforces access control policies (role-based for staff/partners; authenticated status for customers, with potential tiers). Consumes password, session, and lockout policies (potentially from System Configuration).
- **Notes:** Critical for system security and auditability. Ensures robust, performant authentication for all user types. Centralizes access control, distinguishing user types for appropriate permissions and logging.

3.11) System Configuration

- **Purpose:** Manages internal system-level configurations (FallbackPolicy, global parameters) and overall system health parameters not tied to specific external API integrations.
- **Ubiquitous Language:** SystemConfig, FallbackPolicy, SystemParameter, HealthStatus.
- **Key Aggregates:** FallbackPolicy, SystemParameter.
- **Core APIs (Conceptual):**
  - GET /fallback-policies/effective?scope=... : Get active fallback policy.
  - GET /system-parameters/{paramName}: Retrieve a system parameter.
  - GET /system/health: Get aggregated system health.

- **Main Events (Published):**
  - FallbackPolicyActivated (A fallback policy becomes active)
  - SystemParameterUpdated (A global system parameter changes)
- **Related Policies:** Defines system-wide operational policies like FallbackPolicy.
- **Notes:** Ensures consistent behavior during system degradations. Health status relies on other services.

3.12) External Integrations

- **Purpose:** Manages configuration, pre-defined IntegrationContracts, connectivity, and status monitoring for all external third-party hardware and software APIs. Acts as an abstraction and contract enforcement point.
- **Ubiquitous Language:** DeviceIntegration, ExternalAPI, IntegrationPoint, Connectivity, SecretManagement, EndpointConfiguration, IntegrationContract.
- **Key Aggregates:** DeviceConfiguration, IntegrationConfig, IntegrationContract.
- **Core APIs (Conceptual):**
  - GET /integrations/{integrationId}/status: Get status of an integration.
  - GET /integration-contracts/{contractId}: Get contract details.
  - Admin APIs for POST/PUT to manage configurations and contracts.
- **Main Events (Published):**
  - DeviceIntegrationStatusChanged (Connectivity/status of an integrated device changes)
  - IntegrationStatusChanged (Connectivity/status of an external software API changes)
  - IntegrationContractUpdated (Terms of an integration contract change)
- **Related Policies:** Adheres to IntegrationContract terms.
- **Notes:** Crucial for abstracting external dependencies. Manages secret references securely.

3.13) Incident & Enforcement

- **Purpose:** Manages rule violations, operational incidents, equipment faults (FaultReport), and towing workflows. Supports evidence logging, fine issuance, and coordination with enforcement. Handles PlateCloningSuspected alerts and equipment fault reports.
- **Ubiquitous Language:** Incident, Violation, Report, Evidence, Fine, Towing, Warning, TowJob, Misuse, PlateCloningInvestigation, FaultReport, FaultType, FaultResolution.
- **Key Aggregates:** IncidentReport, FaultReport, TowJob.
- **Core APIs (Conceptual):**
  - POST /incidents: Log a new incident/violation.
  - POST /faults: Report a new equipment fault.
  - GET /incidents/{id} / GET /faults/{id}: Retrieve details.
- **Main Events (Published):**

    - IncidentLogged (New incident or violation reported)
    - FaultReported (New equipment fault detected)
    - FaultStatusUpdated (Status of a fault changes)
    - FineIssuedForVehicle (Fine applied as enforcement action)
- **Related Policies (Consumed):** Enforcement escalation policies, FaultBillingAdjustmentPolicy.
- **Notes:** Links faults to equipment. Fault resolution can trigger billing adjustments.

3.14) Analytics & Reporting

- **Purpose:** Consumes data from all operational contexts for historical analysis, trend reporting, financial summaries, and predictive analytics like OccupancyForecast (using a time-series projection engine). Supports strategic decision-making and operational dashboards.
- **Ubiquitous Language:** Report, Dashboard, Metric, Trend, Forecast, Utilization, Revenue, StaffPerformance, OccupancyPrediction, ValidationEffectiveness, ValetOperations, SystemHealthTrend, FaultAnalysis.
- **Key Aggregates:** Read Models (ForecastedOccupancy) and Data Processing Pipelines.
- **Core APIs (Conceptual):**
    - GET /reports/{reportName}?params... : Retrieve pre-defined reports.
    - GET /forecasts/occupancy?lotId=...&targetTime=... : Get occupancy predictions.
- **Main Events (Published):**
    - OccupancyForecastUpdated (New occupancy forecast generated)
- **Related Policies:** Consumes data reflecting policy applications.
- **Notes:** Read-only projections. Forecasting engine is core. Provides data for dynamic pricing, reservation warnings, staffing.

4) Operational Architecture & System Behavior

This section summarizes key platform-wide behaviors and architectural considerations that span multiple microservices, ensuring consistent behavior and system cohesion. It focuses on how services interact and how the system handles overarching concerns

4.1) Communication Patterns

- **Asynchronous (Event-Driven):** The primary mode of inter-service communication to ensure loose coupling and resilience.
  - **Mechanism:** A central Message Broker (e.g., Kafka, RabbitMQ) will be used for publishing and subscribing to domain events.
  - **Event Structure:** Events will have well-defined schemas (e.g., Avro, JSON Schema) managed in a Schema Registry, with versioning to support evolution.
  - **Sample Flow:**
    1. Parking Operations Service publishes VisitReadyForPricing.
    2. Pricing Service consumes this, calculates charges, and publishes PriceQuoteCalculatedForVisit.
    3. Billing Service consumes PriceQuoteCalculatedForVisit, creates a Bill, and publishes BillCreated.
    4. Internal Audit & Event Trace Service subscribes to all these events for logging.
- **Synchronous (Direct API Calls):** Used for request/response interactions where immediate feedback is necessary.
  - **Mechanism:** REST (over HTTP/S) or gRPC (for internal, performance-sensitive calls).
  - **Resilience:** Client-side load balancing, retries, timeouts, and circuit breakers are essential.
  - **Example:** Parking Operations Service might synchronously call Reservations Service to validate a reservation upon vehicle entry.

4.2) Fallback Mode Handling

- **Activation:** Fallback modes are triggered by critical external system failures (e.g., ANPR system offline, payment gateway unavailable) or by manual staff activation. The System Configuration Service manages FallbackPolicy definitions, and the External Integrations Service reports the status of integrated systems.
- **Degraded Mode Support:**
  - **Parking Operations:** Can log VisitLoggedUnderFallback if ANPR fails, allowing manual entry/exit recording. VisitReconciliationJobs are created to process these visits later. Gate operation may switch to manual based on FallbackPolicy.
  - **Billing:** May queue payment processing or offer alternative settlement options if

    the primary payment gateway is down, as defined in the active FallbackPolicy.
    - **Reservations:** May restrict new reservations or operate with cached availability if real-time slot status from Parking Operations is degraded.
- **Key Events:** FallbackPolicyActivated, DeviceIntegrationStatusChanged.

4.3) Observability & Tracing

- **Distributed Trace IDs:** A unique Correlation ID (Trace ID) will be generated at the entry point (e.g., API Gateway) for every external request and propagated through all subsequent synchronous calls (e.g., via HTTP headers) and asynchronous messages (e.g., via message headers).
- **Log/Event Correlation:**
    - All services will use structured logging (e.g., JSON), including the Trace ID and Span IDs in every log entry.
    - The Internal Audit & Event Trace Service will store all domain events, also indexed by Trace ID, allowing for a complete end-to-end view of a business process across multiple services and event hops.
    - Centralized logging (e.g., ELK stack, Splunk) will aggregate logs from all services, enabling searching and correlation by Trace ID.
- **Metrics:** Key operational metrics (request rates, error rates, latencies, queue depths, fault occurrences) will be exposed by each service (e.g., via Prometheus) and aggregated for dashboards (e.g., Grafana).
- **Alerting:** Alerts will be configured based on critical metrics, fault events, and DLQ activity.

4.4) Security & Compliance

- **Role-Based Access Control (RBAC):** Managed by the Staff & User Access Service. Roles define sets of permissions. The API Gateway and individual services enforce authorization based on the authenticated staff user's roles.
- **Customer Authentication:** Typically handled by the API Gateway interacting with an Identity Provider (IdP); Customer Service manages customer profiles but not primary authentication credentials.
- **PCI DSS Boundary:** The Billing Service and its direct integration with the External Integrations Service (for payment gateway communication) are the primary scope for PCI DSS compliance. Tokenization of payment methods is key, with sensitive cardholder data never stored directly by Easy Park Plus systems.
- **GDPR/Data Privacy:** PII is present in Customer, Visit, Reservation, Billing, and potentially Incident & Enforcement contexts. Data minimization, purpose limitation, encryption, and data retention policies (managed by Policy Management or System Configuration) are critical. The Internal Audit & Event Trace Service also needs careful PII handling for logged event payloads.
- **Secrets Management:** API keys, database credentials, and other secrets are managed

via a dedicated secrets vault (e.g., HashiCorp Vault), accessed by services as needed.

4.5) Domain Event Catalog Reference

During implementation a comprehensive Domain Event Catalog will be created and maintained as a separate, living document or in a dedicated schema registry. This catalog will detail:

- Event Name
- Version
- Publishing Service(s)
- Subscribing Service(s)
- Payload Schema (e.g., Avro, JSON Schema)
- Description and Business Purpose
- Idempotency considerations for consumers
- Criticality / SLA expectations (if any)

5) Conclusion & Next Steps

5.1) Conclusion

This microservices architecture for Easy Park Plus offers a strategic path to a scalable, resilient, and adaptable platform. Adhering to Domain-Driven Design and focusing on business process alignment, it establishes clear service boundaries for core operations, system administration, and support functions. The dedicated Internal Audit & Event Trace service enhances transparency and compliance.

The operational architecture—emphasizing event-driven communication, robust fallback mechanisms, comprehensive observability, and security by design—provides a solid foundation. This holistic approach, with clearly defined microservices and operational patterns, will empower Easy Park Plus with greater autonomy via independent deployments, improved scalability, better operational insight through features like fault management and Slot Holds, and a platform aligned with evolving business needs and future innovation in smart urban mobility.

This proposal will serve as the reference for all service modeling and governance decisions. Any changes to responsibilities, schemas, or shared value objects must be reviewed against this model and documented through an approved Architecture Change Request process.

5.2) Next Steps

Should we decide to move forward with this plan the following steps should be taken, adhering to the guiding principles:

1. **Detailed Design & Contract Definition:**
    - Develop precise API contracts (OpenAPI) and event schemas (JSON Schema in a registry) for all services, applying the API-First principle.
    - Finalize and maintain the Domain Event Catalog, detailing event specifics, publishers/subscribers (including Internal Audit & Event Trace), payloads, and idempotency considerations.
2. **Core Functionality & Resilience Implementation:**
    - Implement Fallback Policy logic (managed by System Configuration, consumed by Parking Operations) for degraded operational modes.
    - Define and implement the policy override hierarchy within Policy Management for effective, performant policy resolution.
    - Design detailed state transitions, interactions (synchronous/asynchronous), and error handling for Slot Holds (Reservations) and Fault Management (Incident & Enforcement).
3. **Define and Prioritize Initial MVP Slice:** It is recommended that the MVP focuses on core parking operations, basic customer interactions, and essential financial

transactions. This would likely involve:

- Core Parking Lifecycle
    1. Vehicle entry/exit processing using Parking Operations
    2. Basic ANPR data capture via External Integrations
    3. Vehicle identification through the Vehicle service
- Fundamental Customer Management
    1. Enable simple customer registration and vehicle linking via the Customer service
- Basic Pricing & Billing
    1. Straightforward pricing model in the Pricing service for single visits
    2. Essential payment processing through the Billing service
    3. Integrate with a payment gateway via External Integrations
- Essential Supporting Services
    1. Minimal version of Infrastructure Management for lot configuration
    2. Basic Staff & User Access
    3. Internal Audit & Event Trace to log critical events

4. **Implementation & Operational Readiness:**
    - Establish/update infrastructure (message broker), CI/CD pipelines for independent deployments, and observability tools (logging, metrics, tracing).
    - Implement robust security patterns: RBAC, secure secrets management, and payment tokenization (Billing) for PCI DSS compliance.
5. **Team & Process Adaptation:**
    - Conduct comprehensive team enablement on new domains, service boundaries, ubiquitous language, architectural principles (loose coupling, design for failure), and operational procedures.
    - Establish governance for ongoing architectural evolution, data sovereignty, and change management, treating this plan as a living document.

Appendix

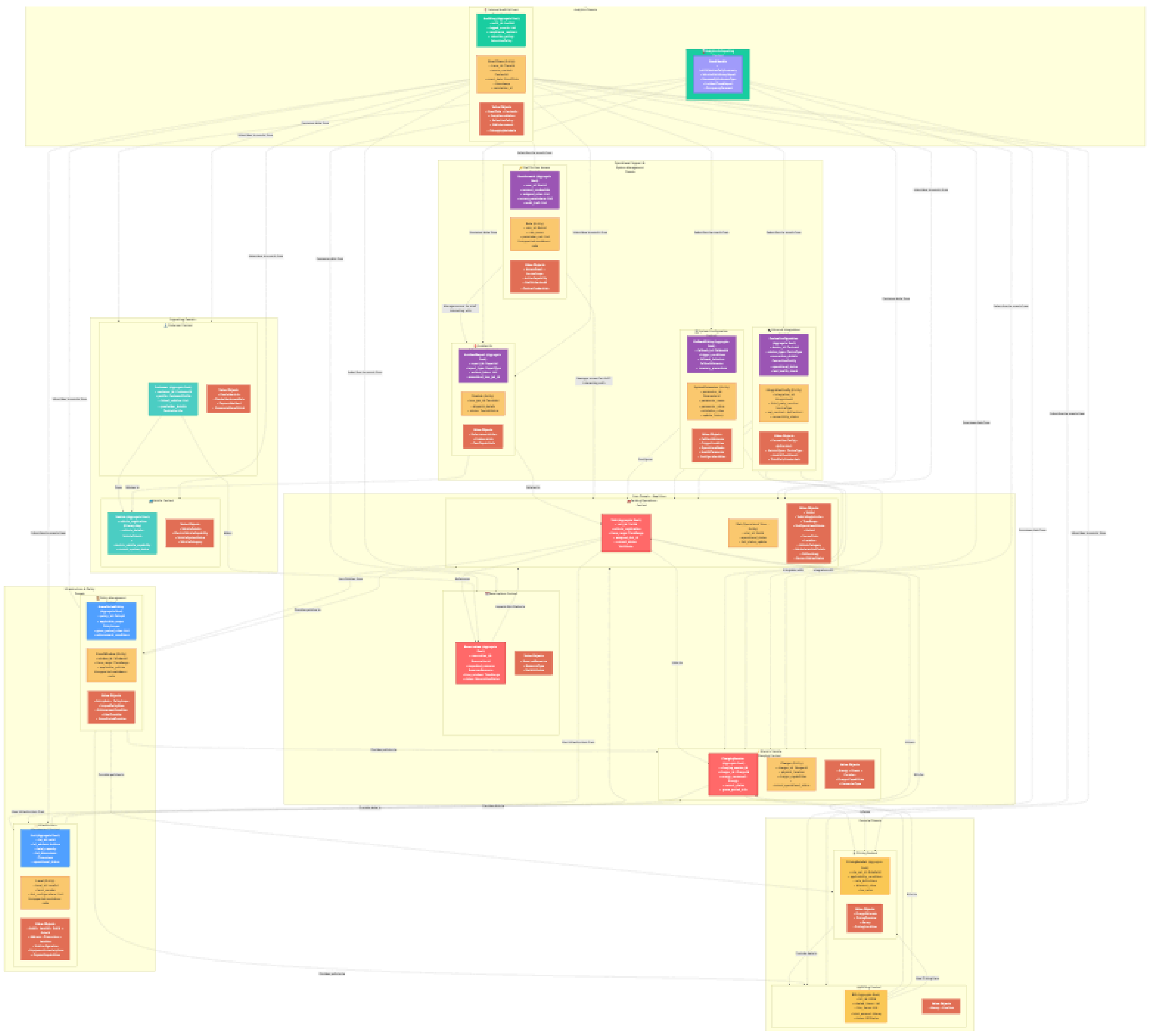
A) Diagrams

This appendix identifies the diagrams created for the proposed architecture. All proposed diagrams are found in the /proposed-ddd-microservices-architecture/diagrams directory of the codebase.

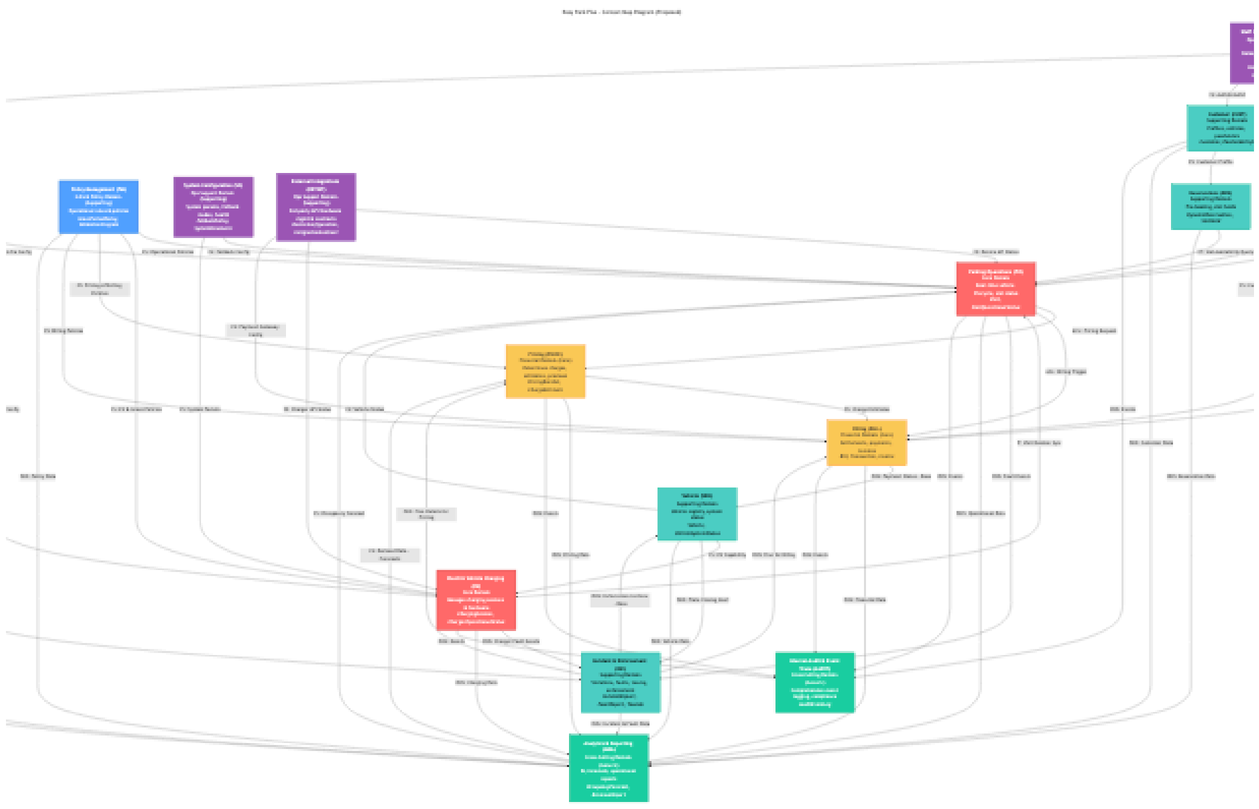
A.1) High-Level System Diagrams

The proposed-ddd-microservices-architecture/diagrams/_system-high-level/ directory contains high-level diagrams in .mermaid and .png formats that provide an overview of the entire system.

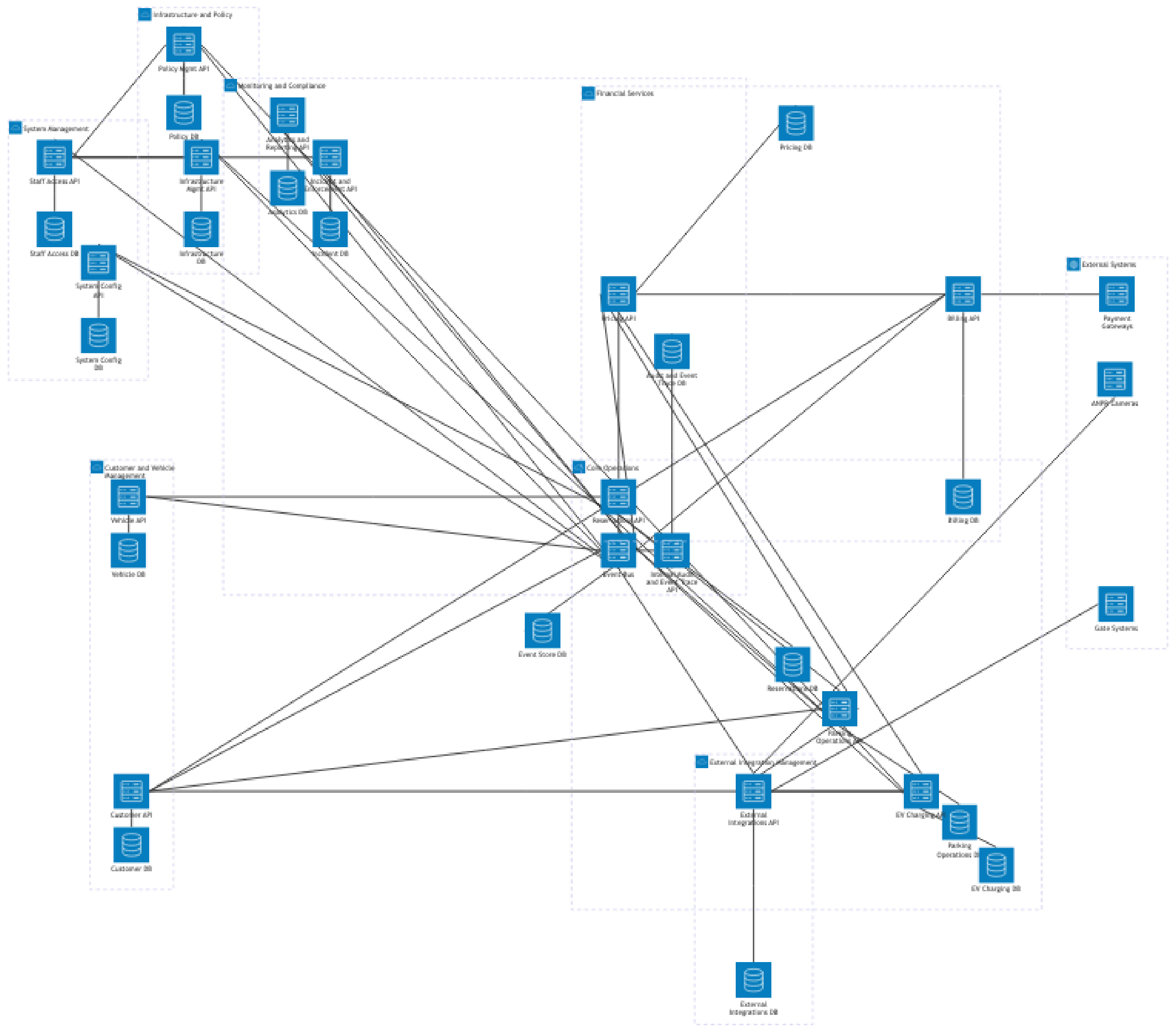
A.1.1) Proposed Bounded Context Diagram



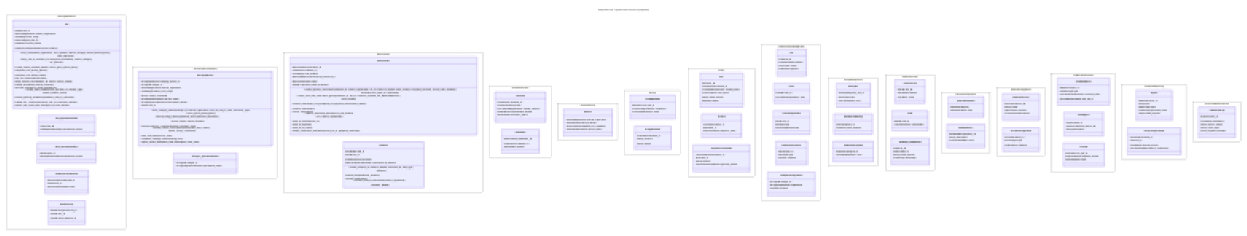
A.1.2) Proposed Context Map Diagram



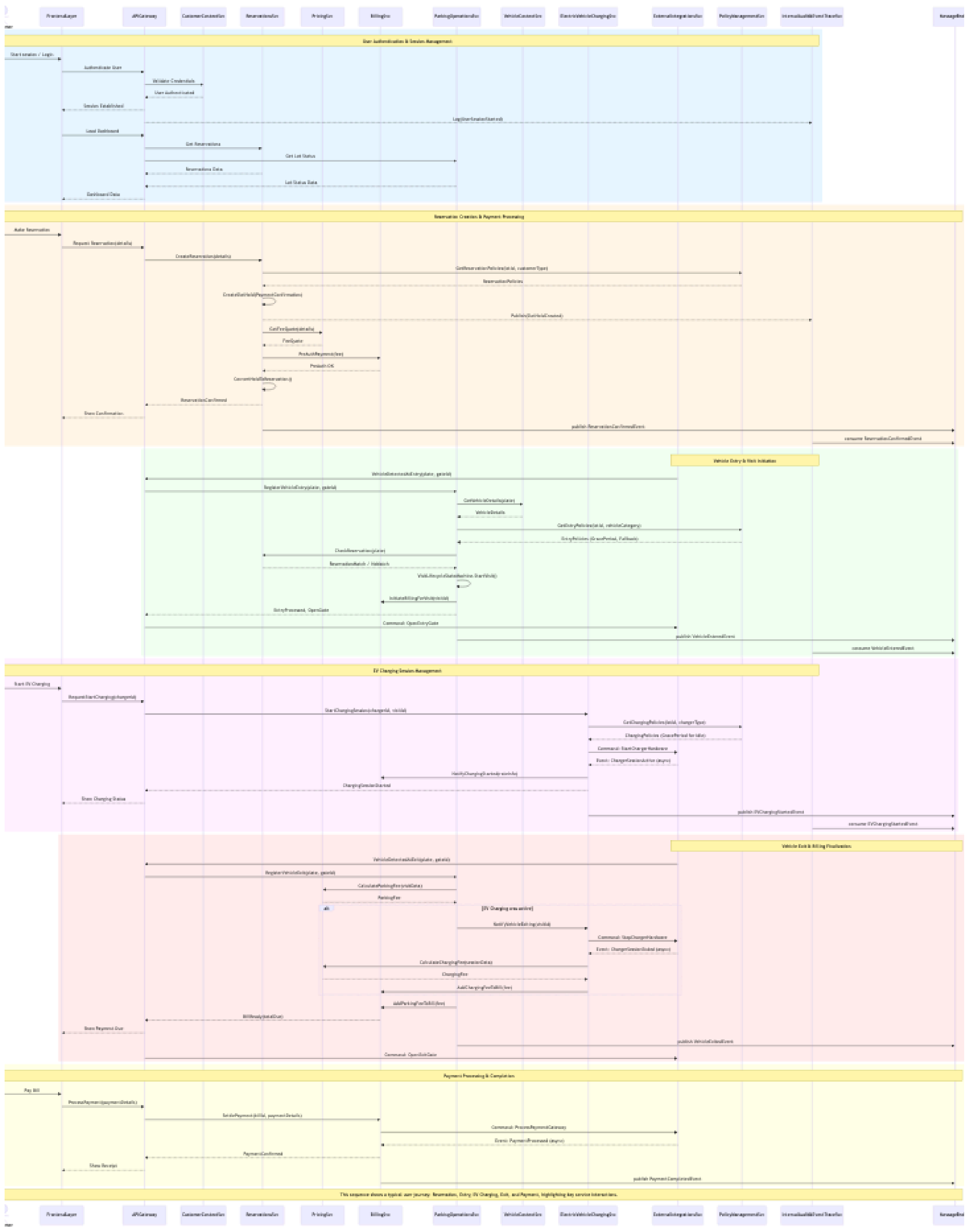
A.1.3) Proposed System Microservices Architecture



A.1.4) Proposed System Class Overview



A1.5) Proposed System Sequence Overview

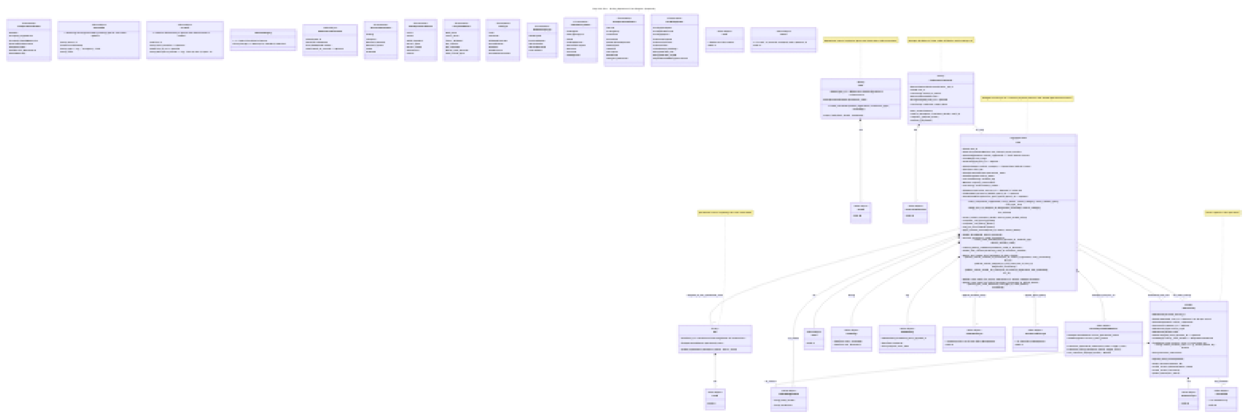

FrontendLayer
APIGateway
CustomerContextSvc
ReservationSvc
PricingSvc
BillingSvc
ParkingOperationsSvc
VehicleContextSvc
ElectricVehicleChargingSvc
ExternalIntegrationSvc
PolicyManagementSvc
InternalAuditAndEventTraceSvc
User Authentication & Session Management
Start session / Login
Authenticate User
Validate Credentials
User Authenticated
Session Established
Log(UserSessionStarted)
Load Dashboard
Get Reservations
Get Lot Status
Reservations Data
Lot Status Data
Dashboard Data
Reservation Creation & Payment Processing
Make Reservation
Request Reservation(details)
CreateReservation(details)
GetReservationPolicies(lotId, customerType)
ReservationPolicies
CreateSlotHold(PaymentConfirmation)
Publish(SlotHoldCreated)
GetFeeQuote(details)
FeeQuote
PreAuthPayment(fee)
PreAuth OK
ConvertHoldToReservation()
ReservationConfirmed
Show Confirmation
publish ReservationConfirmedEvent
consume ReservationConfirmedEvent
Vehicle Entry & Visit Initiation
VehicleDetectedAtEntry(plate, gateId)
RegisterVehicleEntry(plate, gateId)
GetVehicleDetails(plate)
VehicleDetails
GetEntryPolicies(lotId, vehicleCategory)
EntryPolicies (GracePeriod, Fallback)
CheckReservation(plate)
ReservationMatch / NoMatch
Visit.Arrival(StatusMachine.StartVisit)
InitiateBillingForVisit(visitId)
EntryProcessed, OpenGate
Command: OpenEntryGate
publish VehicleEnteredEvent
consume VehicleEnteredEvent
EV Charging Session Management
Start EV Charging
RequestStartCharging(chargerId)
StartChargingSession(chargerId, visitId)
GetChargingPolicies(lotId, chargerType)
ChargingPolicies (GracePeriod for Idle)
Command: StartChargerHardware
Event: ChargerSessionActive (async)
NotifyChargingStarted(visitInfo)
ChargingSessionStarted
Show Charging Status
publish EVChargingStartedEvent
consume EVChargingStartedEvent
Vehicle Exit & Billing Finalization
VehicleDetectedAtExit(plate, gateId)
RegisterVehicleExit(plate, gateId)
CalculateParkingFee(visitData)
ParkingFee
alt
[EV Charging was active]
NotifyVehicleExiting(visitId)
Command: StopChargerHardware
Event: ChargerSessionEnded (async)
CalculateChargingFee(sessionData)
ChargingFee
AddChargingFeeToBill(fee)
AddParkingFeeToBill(fee)
BillReady(totalDue)
Show Payment Due
publish VehicleExitedEvent
Command: OpenExitGate
Payment Processing & Completion
Pay Bill
ProcessPayment(paymentDetails)
SettlePayment(billId, paymentDetails)
Command: ProcessPaymentGateway
Event: PaymentProcessed (async)
PaymentConfirmed
Show Receipt
publish PaymentCompletedEvent
This sequence shows a typical user journey: Reservation, Entry, EV Charging, Exit, and Payment, highlighting key service interactions.

A.2) Other Diagrams

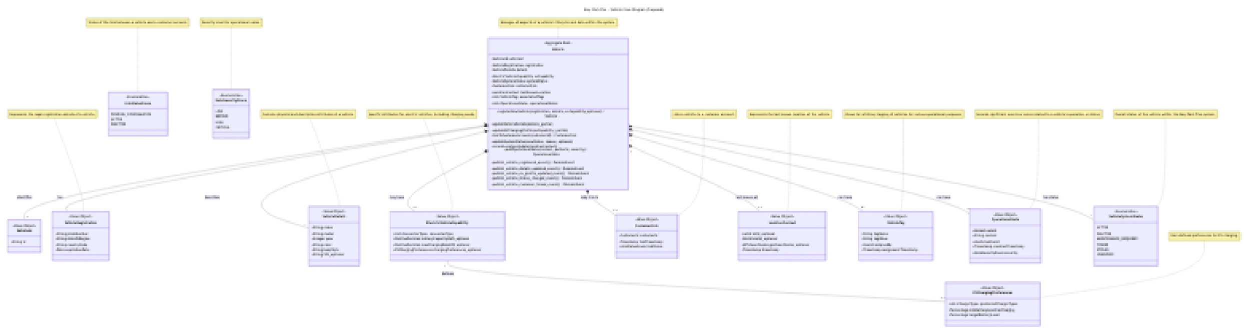
The codebase also includes detailed .mermaid and .png diagrams for:

- **Classes**: proposed-ddd-microservices-architecture/diagrams/class/
- **Communication**: proposed-ddd-microservices-architecture/diagrams/communication/
- **Sequences**: proposed-ddd-microservices-architecture/diagrams/sequence/
- **States**: proposed-ddd-microservices-architecture/diagrams/state/
- **Use-Cases**: proposed-ddd-microservices-architecture/diagrams/use-case/

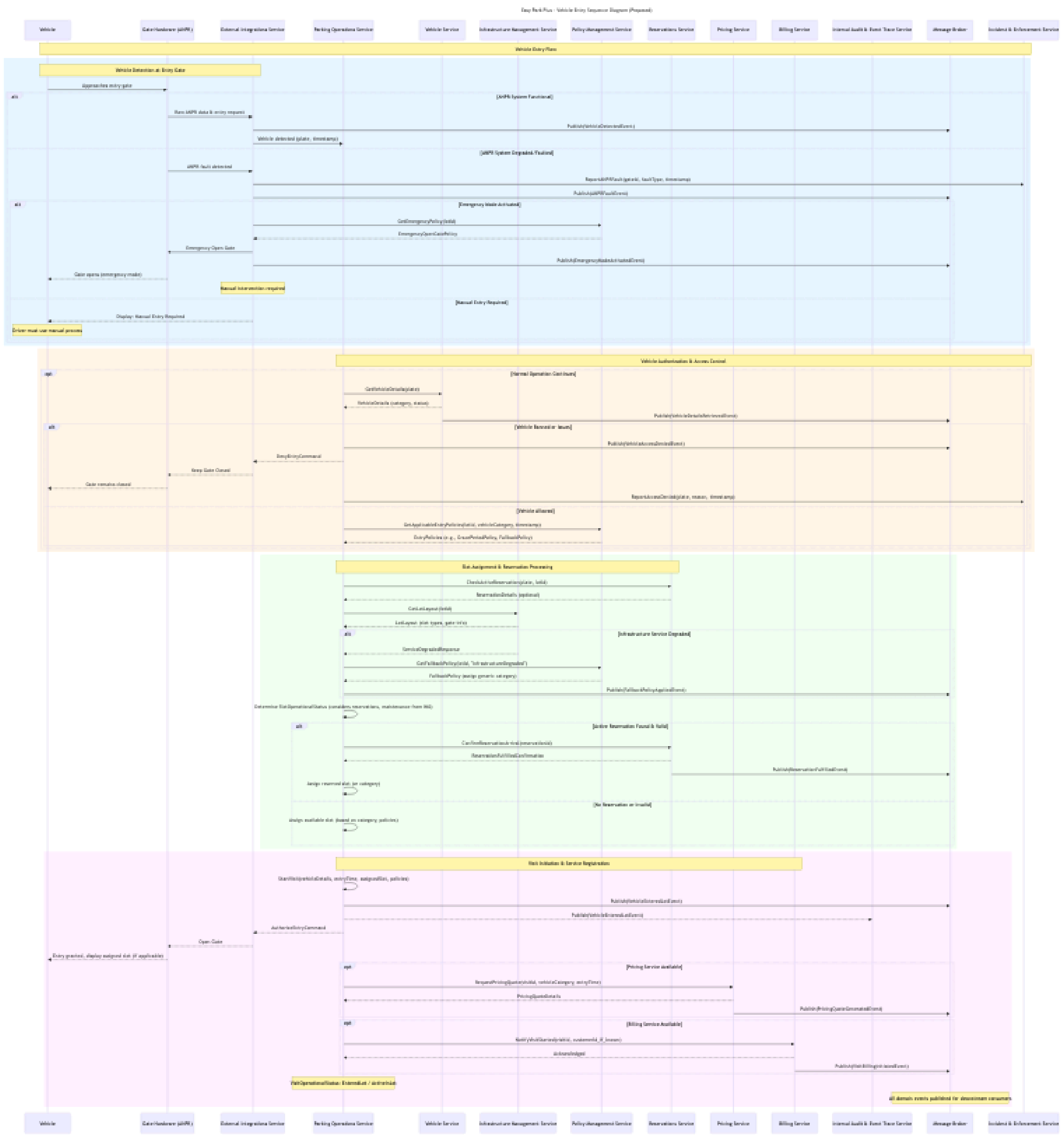
A.2.1) Proposed Parking Operations Class Diagram



A.2.2) Proposed Vehicle Class Diagram

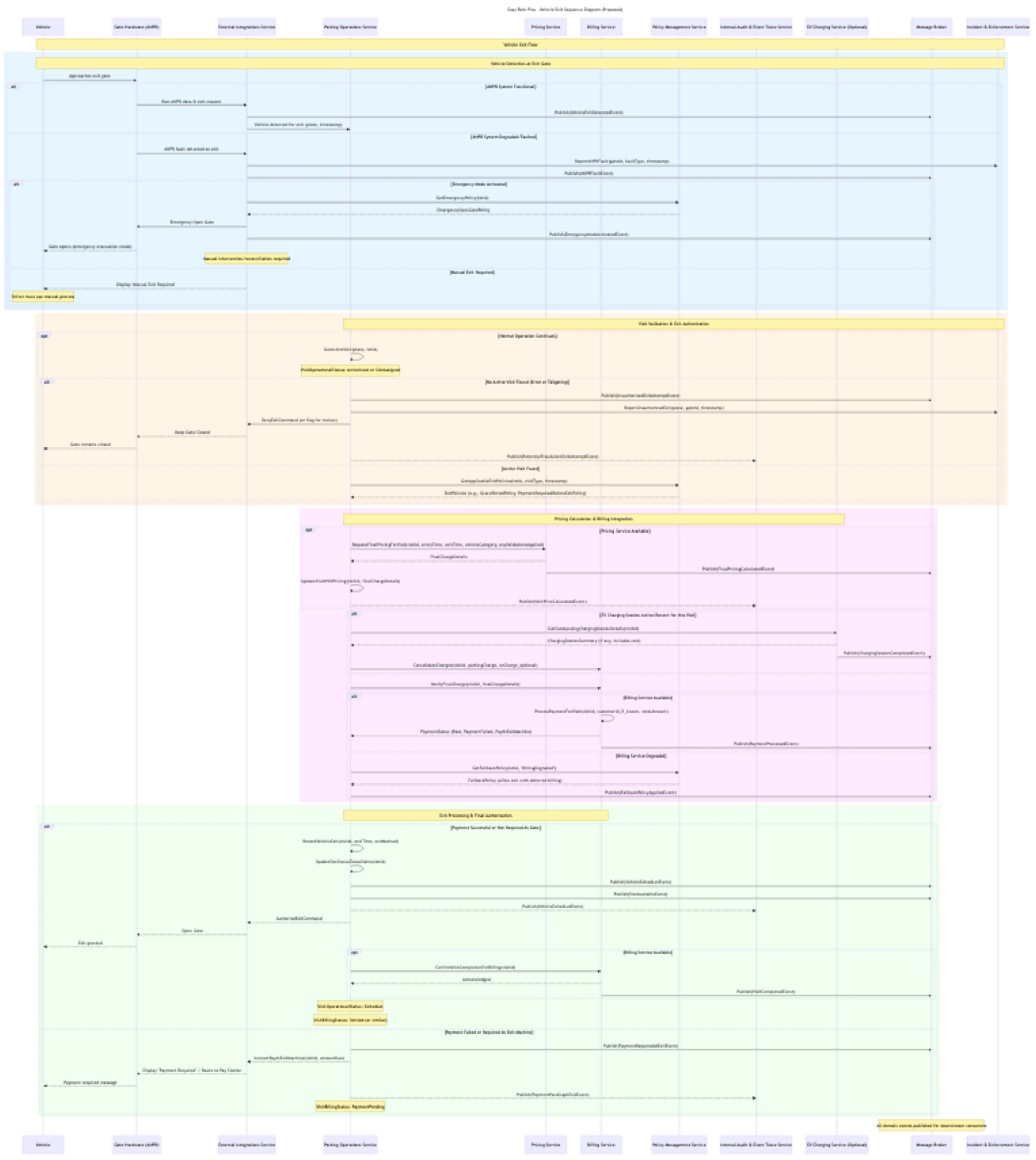


A.2.3) Proposed Vehicle Entry Sequence Diagram

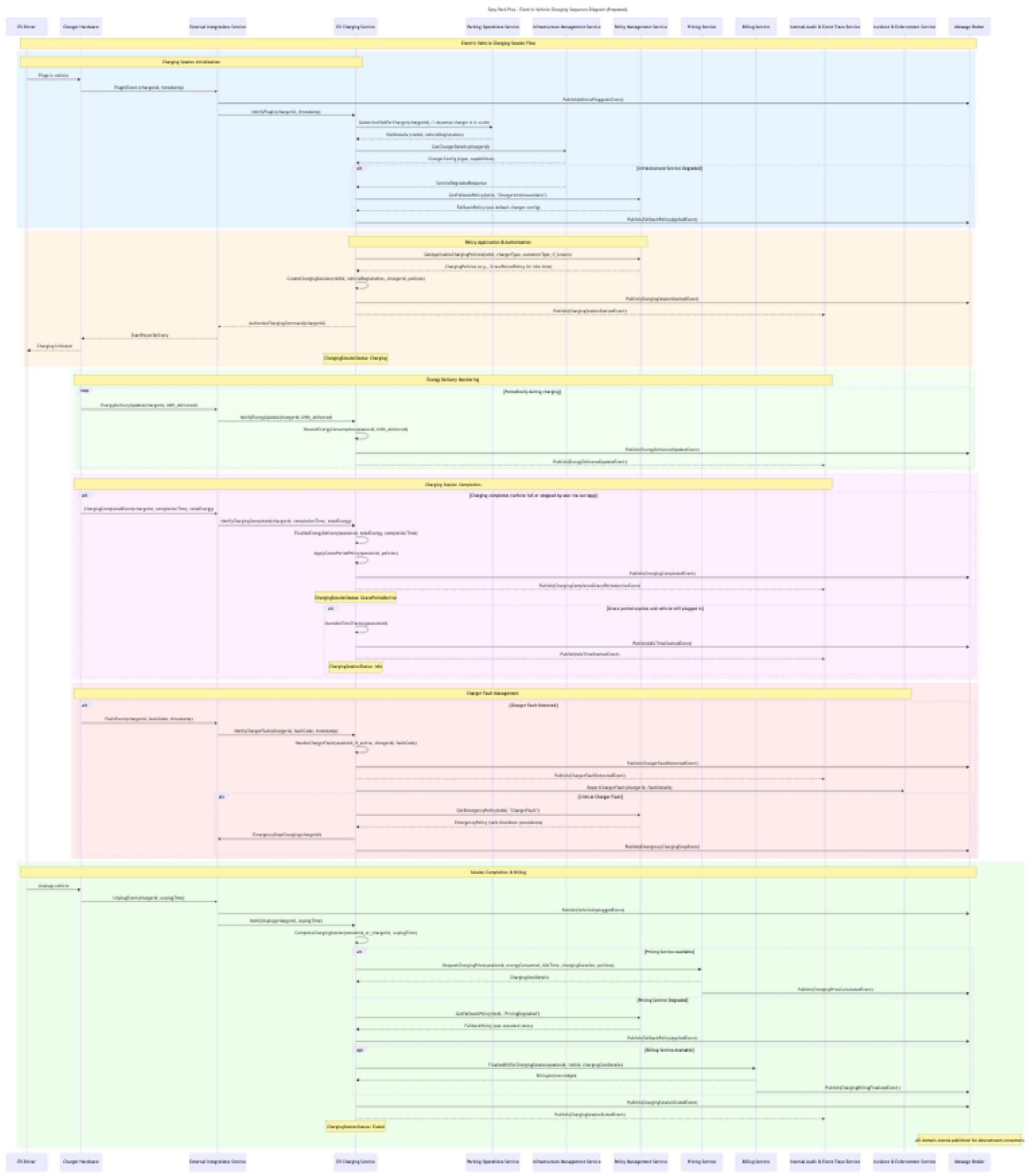


Easy Park Plus - Vehicle Entry Sequence Diagram (Proposed)
Vehicle
Gate Hardware (ANPR)
External Integrations Service
Parking Operations Service
Vehicle Service
Infrastructure Management Service
Policy Management Service
Reservations Service
Pricing Service
Billing Service
Internal Audit & Event Trace Service
Message Broker
Incident & Enforcement Service
Vehicle Entry Flow
Vehicle Detection at Entry Gate
Approaches entry gate
alt
[ANPR System Functional]
Raw ANPR data & entry request
Publish(VehicleDetectedEvent)
Vehicle detected (plate, timestamp)
[ANPR System Degraded/Faulted]
ANPR fault detected
ReportANPRFault(gateId, faultType, timestamp)
Publish(ANPRFaultEvent)
alt
[Emergency Mode Activated]
GetEmergencyPolicy(lotId)
EmergencyOpenGatePolicy
Emergency Open Gate
Publish(EmergencyModeActivatedEvent)
Gate opens (emergency mode)
Manual intervention required
[Manual Entry Required]
Display: Manual Entry Required
Driver must use manual process
Vehicle Authorization & Access Control
opt
[Normal Operation Continues]
GetVehicleDetails(plate)
VehicleDetails (category, status)
Publish(VehicleDetailsRetrievedEvent)
alt
[Vehicle Restrictor Issues]
Publish(VehicleAccessDeniedEvent)
DenyEntryCommand
Keep Gate Closed
Gate remains closed
ReportAccessDenial(plate, reason, timestamp)
[Vehicle Allowed]
GetApplicableEntryPolicies(lotId, vehicleCategory, timestamp)
EntryPolicies (e.g., GracePeriodPolicy, FallbackPolicy)
Slot Assignment & Reservation Processing
CheckActiveReservation(plate, lotId)
ReservationDetails (optional)
GetLotLayout(lotId)
LotLayout (slot types, gate info)
alt
[Infrastructure Service Degraded]
ServiceDegradedResponse
GetFallbackPolicy(lotId, "InfrastructureDegraded")
FallbackPolicy (assign generic category)
Publish(FallbackPolicyAppliedEvent)
Determine SlotOperationalStatus (considers reservations, maintenance from IM)
alt
[Active Reservation Found & Valid]
ConfirmReservationArrival(reservationId)
ReservationFulfilledConfirmation
Publish(ReservationFulfilledEvent)
Assign reserved slot (or category)
[No Reservation or Invalid]
Assign available slot (based on category, policies)
Visit Initiation & Service Registration
StartVisit(vehicleDetails, entryTime, assignedSlot, policies)
Publish(VehicleEnteredLotEvent)
Publish(VehicleEnteredLotEvent)
AuthorizeEntryCommand
Open Gate
Entry granted, display assigned slot (if applicable)
opt
[Pricing Service Available]
RequestPricingQuote(visitId, vehicleCategory, entryTime)
PricingQuoteDetails
Publish(PricingQuoteGeneratedEvent)
opt
[Billing Service Available]
NotifyVisitStarted(visitId, customerId_if_known)
Acknowledged
Publish(VisitBillingInitiatedEvent)
VisitOperationalStatus: EnteredLot / ActiveVisit
All domain events published for downstream consumers

A.2.4) Proposed Vehicle Exit Sequence Diagram


Easy Park Plus - Vehicle Exit Sequence Diagram (Proposed)
Vehicle
Gate Hardware (ANPR)
External Integration Service
Parking Operations Service
Pricing Service
Billing Service
Policy Management Service
Internal Audit & Event Trace Service
EV Charging Service (Optional)
Message Broker
Incident & Enforcement Service
Vehicle Exit Flow
Vehicle Detection at Exit Gate
Approaches exit gate
alt
[ANPR System Functional]
Raw ANPR data & exit request
Vehicle detected for exit (plate, timestamp)
[ANPR System Degraded/Faulted]
ANPR fault detected at exit
[Emergency Mode Activated]
Emergency Open Gate
Gate opens (emergency evacuation mode)
Manual intervention/reconciliation required
[Manual Exit Required]
Display: Manual Exit Required
Driver must use manual process
Visit Validation & Exit Authorization
Pricing Calculation & Billing Integration
Exit Processing & Final Authorization
Open Gate
Exit granted
Payment required message
All domain events published for downstream consumers

A.2.5) Proposed Electric Vehicle Charging Sequence Diagram


Easy Park Plus - Electric Vehicle Charging Sequence Diagram (Proposed)
EV Driver
Charger Hardware
External Integration Service
EV Charging Service
Parking Operations Service
Infrastructure Management Service
Policy Management Service
Pricing Service
Billing Service
Internal Audit & Event Trace Service
Incident & Enforcement Service
Message Broker
Electric Vehicle Charging Session Flow
Charging Session Initialization
Plugs in vehicle
PlugInEvent (chargerId, timestamp)
Publish(VehiclePluggedInEvent)
NotifyPlugIn(chargerId, timestamp)
GetActiveVisitForCharger(chargerId) // assumes charger is in a visit
VisitDetails (visitId, vehicleRegistration)
GetChargerDetails(chargerId)
ChargerConfig (type, capabilities)
alt
[Infrastructure Service Degraded]
ServiceDegradedResponse
GetFallbackPolicy(visitId, 'ChargerInfoUnavailable')
FallbackPolicy (use default charger config)
Publish(FallbackPolicyAppliedEvent)
Policy Application & Authorization
GetApplicableChargingPolicies(visitId, chargerType, customerType_if_known)
ChargingPolicies (e.g., GracePeriodPolicy for idle time)
CreateChargingSession(visitId, vehicleRegistration, chargerId, policies)
Publish(ChargingSessionStartedEvent)
Publish(ChargingSessionStartedEvent)
AuthorizeChargingCommand(chargerId)
StartPowerDelivery
Charging Initiated
ChargingSessionStatus: Charging
Energy Delivery Monitoring
loop
[Periodically during charging]
EnergyDeliveryUpdate(chargerId, kWh_delivered)
NotifyEnergyUpdate(chargerId, kWh_delivered)
RecordEnergyConsumption(sessionId, kWh_delivered)
Publish(EnergyDeliveredUpdateEvent)
Publish(EnergyDeliveredUpdateEvent)
Charging Session Completion
alt
[Charging completes (vehicle full or stopped by user via car/app)]
ChargingCompletedEvent(chargerId, completionTime, totalEnergy)
NotifyChargingComplete(chargerId, completionTime, totalEnergy)
FinalizeEnergyDelivery(sessionId, totalEnergy, completionTime)
ApplyGracePeriodPolicy(sessionId, policies)
Publish(ChargingCompletedEvent)
Publish(ChargingCompletedGracePeriodActiveEvent)
ChargingSessionStatus: GracePeriodActive
alt
[Grace period expires and vehicle still plugged in]
StartIdleTimeTracking(sessionId)
Publish(IdleTimeStartedEvent)
Publish(IdleTimeStartedEvent)
ChargingSessionStatus: Idle
Charger Fault Management
alt
[Charger Fault Detected]
FaultEvent(chargerId, faultCode, timestamp)
NotifyChargerFault(chargerId, faultCode, timestamp)
HandleChargerFault(sessionId_if_active, chargerId, faultCode)
Publish(ChargerFaultDetectedEvent)
Publish(ChargerFaultDetectedEvent)
ReportChargerFault(chargerId, faultDetails)
alt
[Critical Charger Fault]
GetEmergencyPolicy(visitId, 'ChargerFault')
EmergencyPolicy (safe shutdown procedures)
EmergencyStopCharging(chargerId)
Publish(EmergencyChargingStopEvent)
Session Completion & Billing
Unplugs vehicle
UnplugEvent(chargerId, unplugTime)
Publish(VehicleUnpluggedEvent)
NotifyUnplug(chargerId, unplugTime)
CompleteChargingSession(sessionId_or_chargerId, unplugTime)
alt
[Pricing Service Available]
RequestChargingPrice(sessionId, energyConsumed, idleTime, chargingDuration, policies)
ChargingCostDetails
Publish(ChargingPriceCalculatedEvent)
[Pricing Service Degraded]
GetFallbackPolicy(visitId, 'PricingDegraded')
FallbackPolicy (use standard rates)
Publish(FallbackPolicyAppliedEvent)
opt
[Billing Service Available]
FinalizeBillForChargingSession(sessionId, visitId, chargingCostDetails)
BillingActionAcknowledged
Publish(ChargingBillingFinalizedEvent)
Publish(ChargingSessionEndedEvent)
Publish(ChargingSessionEndedEvent)
ChargingSessionStatus: Ended
All domain events published for downstream consumers

B) Infrastructure & Shared Services

This appendix summarizes key infrastructure and shared services supporting the microservices architecture.

B.1) API Gateway

- **Purpose:** Secure entry for external clients; routing, authN/Z, request shaping, BFFs.
- **Key Integration Points:** Client apps (Mobile, Web); Staff & User Access (authN/Z); Backend services (routing).
- **Operational Notes:** Tenant routing, rate limiting, throttling, circuit breaking, SSL, caching. Highly available and scalable.

B.2) Notification Service

- **Purpose:** Centralized alerts and communications (email, SMS, push) to customers/staff.
- **Key Integration Points:** Billing, Reservations, Incident & Enforcement, Parking Ops, EV Charging, System Config, Customer, Staff & User Access (preferences). Consumes events; uses external integrations.
- **Operational Notes:** Templates, retry, opt-outs, delivery tracking.

B.3) System Health Monitor (Conceptual)

- **Purpose:** Aggregates health status of critical system components and integrations.
- **Key Integration Points:** Operations Staff (dashboards); System Configuration; API Gateway (health checks); Alerting.
- **Operational Notes:** Data from external integrations (device/API status), System Config (internal), pings. Publishes SystemDegradedModeActivated events.

B.4) Internal Audit & Event Trace Service

- **Purpose:** Logs domain events (raw payloads/metadata) for compliance, debugging, SLA, traceability.
- **Key Integration Points:** All microservices (write-only); Audit/Compliance tools; Dev/Support tools (restricted).
- **Operational Notes:** High-volume writes, immutable storage, event replay (idempotent consumers), crucial for GDPR/PCI DSS, data retention.

B.5) Retry Queue Manager (Conceptual)

- **Purpose:** Stores failed async events for retrying delivery.
- **Key Integration Points:** Services with critical async calls (External Integrations like Payment/Towing, Notification, Billing).
- **Operational Notes:** Part of message broker (DLQs) or client libs. Tracks retries,

exponential backoff, jitter. Ensures eventual consistency.

B.6) Access Control (RBAC)

- **Purpose:** Enforces authentication and permissions for staff/partner access.
- **Key Integration Points:** API Gateway (all staff/partner); Staff UIs/Admin Tools; Individual services (specific staff permissions).
- **Operational Notes:** Core of Staff & User Access Service. Defines roles/permissions. Centralized identity/access.

B.7) Device/Integration Monitor

- **Purpose:** Tracks operational metrics and connectivity of hardware/external APIs; alerts on faults.
- **Key Integration Points:** System Config (aggregated health); Parking/EV Charging (real-time); External Integrations (data source); Incident & Enforcement (escalation).
- **Operational Notes:** Primarily External Integrations Service. Faults escalate to FaultReports or trigger FallbackPolicy (System Config).

B.8) Message Broker

- **Purpose:** Asynchronous, event-driven communication between microservices.
- **Key Integration Points:** All microservices (publishers/subscribers).
- **Operational Notes:** Choice (Kafka, RabbitMQ) impacts scalability, ordering, stream processing. Highly available/resilient. Manages topics/exchanges.

B.9) Service Discovery

- **Purpose:** Enables dynamic finding of service network locations.
- **Key Integration Points:** Microservices making direct synchronous calls.
- **Operational Notes:** Container orchestration (Kubernetes DNS) or dedicated tools (Consul, Eureka). Essential for scaling/resilience.

B.10) Secrets Management

- **Purpose:** Securely stores and manages access to sensitive information.
- **Key Integration Points:** All services needing secrets, especially External Integrations.
- **Operational Notes:** Dedicated vault (Vault, AWS Secrets Manager). Authentication for retrieval. Strict access policies.

B.11) Centralized Logging

- **Purpose:** Aggregates logs from all microservices for analysis, monitoring, debugging.
- **Key Integration Points:** All microservices (log producers); Ops/Dev teams;

Observability tools (ELK, Splunk, Loki).

- **Operational Notes:** Structured logs with correlation IDs.

B.12) Centralized Metrics

- **Purpose:** Aggregates metrics for performance monitoring, alerting, capacity planning.
- **Key Integration Points:** All microservices (metric producers); Ops/Dev teams; Observability tools (Prometheus, Grafana).
- **Operational Notes:** Collects KPIs, system/business metrics. Drives alerting (Alertmanager).

C) Platform & Execution Strategy

The following stack supports EasyParkPlus's distributed, event-driven architecture. Tools are chosen to ensure resilience, observability, and bounded context autonomy.

C.1) Service Platform

- **Technology:** Kubernetes (e.g., EKS, GKE, AKS)
- **Notes:** Each bounded context is deployed as its own set of stateless services. Container orchestration provides scalability, resilience, and efficient resource management.

C.2) Event Bus

- **Technology:** Apache Kafka (or similar like RabbitMQ, Google Pub/Sub, AWS Kinesis)
- **Notes:** All core contexts publish and subscribe to domain events. Kafka is preferred for high throughput, durability, and stream processing capabilities.

C.3) API Gateway

- **Technology:** Envoy, Kong (or cloud provider solutions like AWS API Gateway, Apigee)
- **Notes:** Performs routing, versioning, authentication/authorization enforcement, rate limiting, and request shaping. Acts as the single entry point for external clients.

C.4) Service Mesh

- **Technology:** Istio (or Linkerd)
- **Notes:** Enables advanced traffic management, mutual TLS (mTLS) for secure service-to-service communication, observability features like distributed tracing propagation, and resilient communication patterns (retries, timeouts, circuit breakers).

C.5) Databases

- **Technology:** PostgreSQL (per service, as a primary choice), MongoDB (for document-centric data like Incidents/Faults), Time-series DB (e.g., InfluxDB for EV energy, metrics), EventStoreDB (for Audit & Trace).
- **Notes:** Each microservice owns its own schema and database instance to ensure loose coupling and independent scalability. Event consumers often use projection models rather than direct cross-service database joins.

C.6) Search Index

- **Technology:** OpenSearch (or Elasticsearch)
- **Notes:** Used by Analytics & Reporting for complex queries, potentially for searching

violations in Incident & Enforcement, or customer lookup in Customer Service.

C.7) Object Storage

- **Technology:** AWS S3 (or Google Cloud Storage, Azure Blob Storage)
- **Notes:** Stores large binary objects like evidence photos/videos for incidents, scanned documents, and potentially long-term audit trails or event archives.

C.8) Authentication / Identity Provider (IdP)

- **Technology:** Auth0 (or Okta, Keycloak, AWS Cognito)
- **Notes:** Provides unified identity management for staff users (via Staff & User Access Service integration) and customers (for client applications). Handles OAuth 2.0 / OIDC flows.

C.9) Observability Suite

- **Technology:** Grafana (visualization), Loki (logs), Jaeger/OpenTelemetry (tracing), Prometheus (metrics).
- **Notes:** Provides a centralized platform for logging, tracing, and metrics aggregation, enabling comprehensive monitoring and debugging of the distributed system.

C.10) Service Deployment Principles

- Bounded contexts are deployed as **independently versioned and scaled services**.
- Services are designed to be **stateless** where possible, running as containerized applications (pods in Kubernetes). Sidecar proxies (e.g., Envoy via Istio) handle cross-cutting concerns like observability and mTLS.
- **Events** are designed to be durable and, where specified (e.g., by Internal Audit & Event Trace service), replayable.
- **Commands** must be idempotent to handle network retries safely.

C.11) Database Strategy

- Each microservice owns its own database schema and, ideally, its own database instance (e.g., a dedicated PostgreSQL instance or schema). This enforces data sovereignty and allows independent scaling and evolution of data models.
- Event consumers (services subscribing to events from other services) build and maintain their own **projection models** (read models) of the data they need. Direct cross-service database joins are strictly avoided.
- The Analytics & Reporting service aggregates data from various domain events into its own data warehouse (e.g., OpenSearch for real-time dashboards, Snowflake/BigQuery for batch reporting) for analytical queries.

C.12) Governance

- **Schema Evolution:** All Domain Event schemas and public API schemas will be versioned (e.g., using Semantic Versioning). Event schemas (e.g., Avro, Protobuf) will be managed in a central Schema Registry.
- **Backward Compatibility:** Services consuming events or APIs must be designed to be backward compatible with at least N-1 versions, allowing staggered deployments and reduced coupling.
- **Change Management Process:**
  - Minor API/Event changes (non-breaking, additive) can follow a lightweight review.
  - Major (breaking) changes to public APIs or critical Domain Event schemas require a formal Request for Comments (RFC) process, review by an architecture board or relevant stakeholders, and comprehensive regression testing, including contract testing between services.
- **API Design Guidelines:** A common set of API design guidelines (e.g., RESTful principles, naming conventions, error handling) will be established and enforced.
- **Technology Radar:** Maintain a technology radar to guide adoption of new technologies and deprecation of old ones in a coordinated manner.
- **Regular Architecture Reviews:** Periodically review the architecture to identify areas for improvement, address technical debt, and re-align with evolving business goals.

D) Future Considerations

This appendix outlines potential future capabilities and their architectural implications.

D.1) Bicycle Parking

- **Description:** Dedicated secure zones or racks for bicycle parking, potentially with its own reservation and billing model.
- **When to Prioritize:** Market demand for urban micro-mobility solutions, city partnerships, sustainability initiatives.
- **Likely System Dependencies:**
    - Infrastructure Management: BikeStorageZone as a new type of area/equipment.
    - Reservations Service: If reservable.
    - Parking Operations: Usage tracking if integrated with general access.
    - Customer Service: Linking bike parking passes or access to customer accounts.
    - Billing Service: Specialized billing or as part of a broader mobility pass.
    - Policy Management: Defining access rules, pricing policies for bike parking.
- **Notes on Modeling Impact:**
    - New BikeStorageZone entity in Infrastructure Management.
    - Potentially new BikeReservation type in Reservations.
    - New BikeVisit or similar in Parking Operations if tracking individual usage.
    - New policy types in Policy Management.

D.2) Alternative Vehicle Storage & Services

- **Description:** Designated areas for parking/charging e-scooters or other micro-mobility vehicles, potentially in partnership with third-party providers.
- **When to Prioritize:** Partnerships with micro-mobility operators, city regulations encouraging multi-modal transport, customer demand.
- **Likely System Dependencies:**
    - Infrastructure Management: MicroMobilityZones, charging points for these
    - Policy Management: Rules for access, usage, and provider policy integration.
    - Parking Operations: If Easy Park Plus staff manage these zones or their usage.
    - External Integrations: If micro-mobility provider APIs are involved
    - Billing Service: If Easy Park Plus facilitates or charges for these services.
- **Modeling Impact:**
    - New zone types in Infrastructure Management.
    - New integration contracts and configurations in External Integrations.
    - Potential new MicroMobilityPolicy types in Policy Management.

D.3) Fleet & Delivery Logistics Integration

- **Description:** Specialized services for fleet vehicles (e.g., delivery vans, corporate car shares), including designated loading/unloading zones, pre-arranged access, and consolidated billing.
- **When to Prioritize:** B2B partnerships with logistics companies, demand for urban consolidation centers, desire to optimize loading zone usage.
- **Likely System Dependencies:**
  - Customer Service: New FleetAccount or enhanced BusinessCustomer type with fleet-specific attributes.
  - Parking Operations: FleetVisit type, specialized slot assignment for loading zones
  - Policy Management: FleetAccessPolicy, FleetBillingPolicy
  - Billing Service: Consolidated invoicing, per-vehicle or per-account billing
  - Infrastructure Management: Defining LoadingBay or FleetStagingZone types.
  - Reservations Service: Reserving loading zones or blocks of time for fleets.
- **Modeling Impact:**
  - New FleetAccount entity or attributes in Customer.
  - New FleetVisit attributes or type in Parking Operations.
  - Specialized policy types in Policy Management.
  - Enhancements to Invoice aggregation in Billing.

D.4) Smart Parcel Lockers

- **Description:** Offering secure parcel locker services within parking facilities, potentially integrated with delivery company APIs.
- **When to Prioritize:** Diversification of revenue streams, partnerships with e-commerce/delivery companies, enhancing convenience for parkers.
- **Likely System Dependencies:**
  - Infrastructure Management: Modeling ParcelLockerBank as an EquipmentInventoryItem and its location.
  - Policy Management: Access rules for lockers
  - External Integrations: API integration with locker provider systems or delivery company APIs for booking, access codes, and status.
  - Customer Service: Potentially linking locker usage to customer accounts.
  - Billing Service: If there's a fee for locker usage or extended storage.
- **Modeling Impact:**
  - New EquipmentInventoryItem type for ParcelLockerBank.
  - New IntegrationContract and IntegrationConfig for locker APIs.
  - Potentially new policy types for locker access and fees.

D.5) Vehicle-to-Grid Charging

- **Description:** Allowing Elective Vehicles to not only charge but also discharge power back to the grid or building, with potential for customers to earn credits.
- **When to Prioritize:** Energy market developments, utility partnerships, advanced EV

charger capabilities becoming widespread, customer interest in Vehicle-to-Grid programs.
- **Likely System Dependencies:**
    - Electric Vehicle Charging Service: Major enhancements to ChargingSession to track bi-directional energy flow, manage discharge schedules, and interact with Vehicle-to-Grid-capable chargers.
    - Pricing Service: New rate structures for energy sold back to the grid, dynamic pricing based on grid demand/supply.
    - Billing Service: Handling energy credits, debits, and potentially complex settlements with customers and utility providers.
    - Infrastructure Management: EquipmentInventoryItem for chargers needs to specify Vehicle-to-Grid capability.
    - External Integrations: APIs for Vehicle-to-Grid-capable chargers, utility demand-response programs, and energy market pricing.
    - Policy Management: Policies for participation eligibility, discharge limits, compensation rates.
- **Modeling Impact:**
    - Significant changes to ChargingSession and Charger models.
    - New EnergyFlowDirection attribute.
    - Complex new pricing and billing rules.
    - New policy types related to V2G participation and compensation.

D.6) Dynamic Curbside Management / Municipal APIs

- **Description:** Integrating with city systems to manage or reserve curbside spaces for short-term parking, loading/unloading, or ride-hail pickups/drop-offs.
- **When to Prioritize:** City partnerships, smart city initiatives, demand for better curbside utilization, regulations requiring such integration.
- **Likely System Dependencies:**
    - Reservations Service: Booking/managing curbside slot reservations if applicable.
    - Parking Operations: Monitoring and potentially enforcing curbside usage
    - Policy Management: Specific curbside rules, time limits, vehicle type restrictions.
    - External Integrations: APIs with municipal systems (availability, permits, payment).
    - Infrastructure Management: Model curbside zones managed by Easy Park Plus.
    - Pricing & Billing: Updates to facilitate payment for municipal curbside usage.
- **Modeling Impact:**
    - Potentially new CurbsideZone entity in Infrastructure Management.
    - New IntegrationContract for municipal APIs.
    - New policy types for curbside regulations.
    - Reservations might need to handle new resource types.

D.7) Valet Robotization or Automated Parking Systems

- **Description:** Integration with robotic valet systems or fully automated parking garages where vehicles are moved and parked by machines.
- **When to Prioritize:** New lot acquisitions with Automated Parking Systems, partnerships with vendors, desire for higher density parking.
- **Likely System Dependencies:**
    - Parking Operations: Significant changes to visit and slot assignment logic; interaction with Automated Parking System control system instead of manual valet activity. New AutomatedParkingProcess or similar.
    - Infrastructure Management: Modeling the Automated Parking System as a specialized EquipmentInventoryItem or a new AutomatedParkingSystem entity with unique layout and constraints.
    - External Integrations: API for command and control (request park, etc.).
    - Incident & Enforcement: Handling faults or incidents within the System.
- **Modeling Impact:**
    - Potentially a new AutomatedVisit type or significant changes to Visit and Slot interactions.
    - New complex IntegrationContract for Automated Parking Systems.
    - Safety and fault handling policies in Policy Management would be critical.

D.8) Advanced Customer Programs

- **Description:** Multi-city passes, corporate cross-lot pooling agreements, loyalty and rewards programs (e.g., discounted rates after N visits), API access for corporate partners to manage their users/vehicles.
- **When to Prioritize:** Business development initiatives, customer retention strategies, desire to increase B2B revenue.
- **Likely System Dependencies:**
    - Customer Service: Enhancements to Customer and PassholderInfo to support program enrollment, loyalty tiers, points accumulation.
    - Policy Management: Rules for loyalty point earning/redemption, conditions for multi-city pass usage, corporate account access rules.
    - Pricing Service: Applying loyalty discounts or special program rates.
    - Billing Service: Reflecting loyalty redemptions on bills, managing complex corporate billing for pooled resources.
    - API Gateway: Exposing new APIs for corporate partners.
    - Analytics & Reporting: Tracking program effectiveness and customer engagement.
- **Modeling Impact:**
    - New LoyaltyProgram, LoyaltyTier, PointsAccount entities likely in Customer Service.
    - New policy types in Policy Management.
    - Pricing rules to incorporate loyalty discounts.

D.9) Real Estate or Zoning Metadata

- **Description:** Storing and utilizing detailed real estate information for lots (e.g., ownership, lease terms) and zoning compliance data (e.g., environmental constraints, residential vs. commercial use restrictions, proximity to sensitive structures like schools/hospitals).
- **When to Prioritize:** Expansion into new cities with complex zoning, managing a diverse property portfolio, compliance requirements.
- **Likely System Dependencies:**
    - Infrastructure Management: Adding new attributes to the Lot and potentially Level entities to store this metadata.
    - Policy Management: Rules in policies might become conditional based on this zoning or real estate metadata (e.g., certain operational hours only allowed in specific zones).
    - Analytics & Reporting: For reporting on property portfolio and compliance.
- **Modeling Impact:**
    - Extension of Lot and Level aggregates/entities in Infrastructure Management.
    - Potentially new conditions for policy evaluation in Policy Management.