Product Requirements Document (PRD)

Autonomous Vehicle Predictive Maintenance Platform

Version: 1.0
Last Updated: October 13, 2025
Project Code: AVPM-2025

Executive Summary

Vision

Build an AI-powered autonomous predictive maintenance platform for vehicles that prevents breakdowns before they occur, autonomously schedules service appointments, and creates a closed feedback loop to manufacturing for continuous quality improvement.

Core Value Proposition

- **For Vehicle Owners:** Peace of mind through proactive maintenance, reduced breakdown incidents by 93%, and personalized trip planning with safety checks
- **For Service Centers:** 50% improvement in utilization through predictable scheduling and optimized workload distribution
- **For Manufacturers:** 31% reduction in defect rates through automated RCA/CAPA analysis and manufacturing feedback loops

Success Metrics

Metric	Current State	Target State	Improvement
Breakdown-related visits	68%	5%	-93%
Service center utilization	52%	78%	+50%
Customer NPS	42	68	+62%

Warranty claims (per 1000 vehicles)	87	59	-32%
Manufacturing defect rate (ppm)	420	290	-31%

---

1. SYSTEM ARCHITECTURE OVERVIEW

1.1 High-Level Architecture

```
┌──────────────────────────────────────────────────────────┐
│                 MASTER AGENT                              │
│        (Orchestrator + UEBA Security)                     │
└──────────────────────────────────────────────────────────┘
        │                        │
┌───────▼────────────────┐  ┌───▼──────────────────────┐
│ PREDICTIVE PIPELINE    │  │ CUSTOMER INTERACTION     │
│ • Data Analysis Agent  │  │ • Engagement Agent       │
│ • Diagnosis Agent      │  │ • Scheduling Agent       │
│ • Quality Insights Module │ │ • Feedback Agent       │
└────────────────────────┘  └──────────────────────────┘
        │                        │
        └────────────┬───────────┘
                     │
      ┌──────────────▼──────────────┐
      │ DATA INFRASTRUCTURE LAYER   │
      │ • TimescaleDB (Telemetry)   │
      │ • PostgreSQL (Relational)   │
      │ • Kafka (Streaming)         │
      │ • Redis (Cache/State)       │
      │ • S3 (Assets/Models)        │
      └─────────────────────────────┘
```

1.2 Core System Principles

1. **Agent Autonomy with Oversight:** Each worker agent operates independently within its domain; Master Agent coordinates and ensures security
2. **Fail-Safe Architecture:** Graceful degradation with human escalation paths
3. **Explainable AI:** Every prediction includes confidence scores and contributing factors

4. **Privacy-First:** AES-256 encryption at rest, TLS 1.3 in transit, GDPR/DPDP compliant
5. **Horizontal Scalability:** Architecture scales from 1,000 to 1,000,000 vehicles without changes

1.3 Technology Stack

Backend:

- Python 3.11+ (FastAPI for APIs, asyncio for concurrency)
- Node.js 20+ (Worker agents requiring real-time communication)
- Go (High-performance scheduling optimization)

Data Layer:

- TimescaleDB (time-series telemetry data)
- PostgreSQL 15+ (relational data)
- Apache Kafka (event streaming)
- Redis 7+ (caching, state management, rate limiting)
- S3/MinIO (object storage)

ML/AI:

- TensorFlow/PyTorch (deep learning models)
- XGBoost/LightGBM (tabular data)
- Scikit-learn (preprocessing, traditional ML)
- SHAP (explainability)
- MLflow (model registry and versioning)

Frontend:

- Next.js 14 (App Router, React Server Components)
- React 18
- React Three Fiber (R3F) + Three.js (3D visualization)
- Tailwind CSS + shadcn/ui
- Recharts/ECharts (data visualization)
- Zustand (state management)

Infrastructure:

- Kubernetes (container orchestration)
- Docker (containerization)
- AWS/GCP/Azure (cloud platform)
- Terraform (IaC)
- GitHub Actions (CI/CD)

Communication:

- Twilio (Voice, SMS)
- Google Cloud TTS (text-to-speech)
- Whisper AI (speech-to-text)
- GPT-4/Claude (conversational AI)
- Firebase Cloud Messaging (push notifications)

---

2. BACKEND ARCHITECTURE

2.1 Service Architecture

The backend follows a microservices architecture with the following core services:

2.1.1 Ingestion Service

Purpose: Receive and validate vehicle telemetry data

Responsibilities:

- Accept telemetry data via REST API and MQTT
- Validate data schemas and checksums
- Perform initial data quality checks
- Publish validated data to Kafka topics
- Handle rate limiting and backpressure

Technical Specifications:

- **Language:** Python (FastAPI) + asyncio for high concurrency
- **Input Formats:** JSON over HTTPS, MQTT binary payloads
- **Throughput Target:** 100,000 events/second
- **Latency Target:** <50ms p99
- **Data Validation:**
  - Schema validation using Pydantic models
  - VIN format validation (ISO 3779)
  - Timestamp monotonicity checks
  - Sensor value range validation
  - GPS coordinate validation

API Endpoints:

```
POST /api/v1/telemetry/ingest
POST /api/v1/telemetry/batch
GET  /api/v1/telemetry/health
```

Data Flow:

Vehicle → [HTTPS/MQTT] → Ingestion Service → Kafka Topic → Stream Processor

2.1.2 Stream Processing Service

Purpose: Real-time feature engineering and anomaly detection

Responsibilities:

- Consume telemetry from Kafka
- Compute rolling window features (1min, 5min, 1hr, 24hr windows)
- Detect statistical anomalies
- Enrich data with vehicle metadata
- Write processed features to TimescaleDB
- Publish anomalies to diagnosis queue

Technical Specifications:

- **Language:** Python (with Kafka Streams or Flink for complex CEP)
- **Processing Model:** Event-time processing with watermarks
- **State Management:** RocksDB-backed state stores
- **Windowing:** Sliding and tumbling windows
- **Checkpoint Interval:** 10 seconds

Feature Engineering Pipeline:

1. **Raw Data Cleaning:**
    - Remove duplicates (dedup by VIN + timestamp)
    - Interpolate missing values (linear for <5 consecutive)
    - Apply median filter for accelerometer noise
    - Z-score clipping for outliers (±4σ)

Rolling Window Features (per component):

Brake System:
- brake_pad_thickness_mean_7d
- brake_pad_wear_rate_30d (mm/1000km)
- hard_brake_count_24h
- brake_pressure_max_1h
- brake_temp_rolling_std_24h

Battery:
- battery_voltage_mean_24h

- voltage_drop_during_crank
- charge_acceptance_rate
- deep_discharge_events_7d
- internal_resistance_estimate

Engine:
- coolant_temp_max_24h
- oil_pressure_min_7d
- rpm_variance_1h
- throttle_position_avg_24h
- fuel_trim_deviation

Transmission:
- gear_shift_smoothness_score_7d
- transmission_temp_rolling_mean_24h
- clutch_slip_events_30d
- shift_time_p95_7d

2.
3. **Derived Features:**
   - Driving pattern classification (urban/highway/aggressive)
   - Load estimation from suspension compression
   - Weather impact score from ambient temp + location
   - Maintenance urgency score

2.1.3 Model Inference Service

Purpose: Serve ML models for failure prediction

Responsibilities:

- Load and serve multiple model versions
- Perform batch and real-time inference
- Compute SHAP explainability values
- Handle A/B testing of model versions
- Monitor model performance and drift

Technical Specifications:

- **Language:** Python (FastAPI) with TensorFlow Serving / Triton
- **Model Formats:** SavedModel, ONNX, XGBoost native
- **Inference Modes:**
  - Real-time (single vehicle, <100ms latency)
  - Batch (1000s of vehicles, <5 min)

- **GPU Support:** Optional NVIDIA T4/V100 for deep learning models
- **Model Versioning:** Blue-green deployment strategy

Supported Models:

1. **Brake System Failure Predictor**
    - Architecture: LSTM (64→32 units) + Dense layers
    - Input: 30-day window of 9 features
    - Output: Failure probability (next 15 days), confidence interval
    - Training Data: 8,400 historical failures
    - Accuracy: 92.3%
2. **Battery Health Predictor**
    - Architecture: XGBoost (200 trees, max_depth=10)
    - Input: 6 battery health features
    - Output: State of Health (0-100%), Remaining Useful Life (days)
    - Training Data: 12,200 failures
    - Accuracy: 89.7%
3. **Transmission Predictor**
    - Architecture: Ensemble (Random Forest + Neural Network)
    - Input: 5 transmission features
    - Output: Failure probability, failure mode classification
    - Training Data: 3,100 failures
    - Accuracy: 87.2%
4. **Engine Predictor**
    - Architecture: Gradient Boosted Trees
    - Input: 8 engine health features
    - Output: Component-level risk scores
    - Training Data: 4,700 failures
    - Accuracy: 90.1%

Model Serving API:

```
POST /api/v1/models/predict
POST /api/v1/models/batch-predict
GET  /api/v1/models/{model_name}/metadata
POST /api/v1/models/explain (SHAP values)
```

2.1.4 Master Agent Orchestration Service

Purpose: Central coordinator for all AI agents

Responsibilities:

- Monitor all data streams and agent status
- Route tasks to appropriate worker agents
- Enforce security policies via UEBA
- Handle escalations and edge cases
- Maintain vehicle state machines
- Generate executive dashboards

Technical Specifications:

- **Language:** Python (FastAPI with asyncio)
- **Message Queue:** RabbitMQ with dead-letter queues
- **State Storage:** Redis (vehicle state cache)
- **Orchestration Pattern:** Saga pattern for distributed transactions
- **Concurrency:** AsyncIO with worker pool (50-100 concurrent tasks)

State Machine Per Vehicle:

```
States:
- HEALTHY (no issues detected)
- MONITORING (anomaly detected, confidence <70%)
- PREDICTION_CONFIRMED (failure predicted, confidence ≥85%)
- CUSTOMER_CONTACTED (engagement initiated)
- APPOINTMENT_SCHEDULED
- SERVICE_IN_PROGRESS
- SERVICE_COMPLETED
- FEEDBACK_COLLECTED

Transitions:
HEALTHY → MONITORING (anomaly detected)
MONITORING → PREDICTION_CONFIRMED (diagnosis confirms issue)
MONITORING → HEALTHY (false alarm, confidence drops)
PREDICTION_CONFIRMED → CUSTOMER_CONTACTED (engagement agent called)
CUSTOMER_CONTACTED → APPOINTMENT_SCHEDULED (customer confirmed)
APPOINTMENT_SCHEDULED → SERVICE_IN_PROGRESS (check-in at service center)
SERVICE_IN_PROGRESS → SERVICE_COMPLETED (work order closed)
SERVICE_COMPLETED → FEEDBACK_COLLECTED (survey submitted)
FEEDBACK_COLLECTED → HEALTHY (cycle reset)
```

Task Routing Logic:

```
def route_task(prediction_result):
    if prediction_result['confidence'] >= 0.85:
```

```
        # High confidence: Proceed with customer engagement
        publish_to_queue('customer_engagement', {
            'vehicle_id': prediction_result['vehicle_id'],
            'component': prediction_result['component'],
            'days_to_failure': prediction_result['days_to_failure'],
            'priority': classify_priority(prediction_result),
            'customer_profile': get_customer_profile(vehicle_id)
        })
    elif prediction_result['confidence'] >= 0.70:
        # Medium confidence: Enhanced monitoring
        schedule_enhanced_monitoring(prediction_result['vehicle_id'])
    else:
        # Low confidence: Log for pattern analysis
        log_low_confidence_prediction(prediction_result)
```

UEBA Integration:

- Every agent action passes through UEBA check before execution
- Anomaly scores >75 block action and alert security team
- Anomaly scores >90 disable agent immediately

2.1.5 Worker Agent Services

A. Data Analysis Agent

Purpose: Continuous monitoring and early warning detection

Responsibilities:

- Subscribe to Kafka telemetry stream
- Compute features in real-time
- Detect statistical anomalies
- Publish alerts to diagnosis queue

Implementation Details:

- **Deployment:** Kubernetes StatefulSet (for state persistence)
- **Scaling:** Horizontal (partition Kafka topics by VIN hash)
- **Anomaly Detection Methods:**
    - Statistical: Z-score, IQR, Grubbs' test
    - ML-based: Isolation Forest, One-Class SVM
    - Time-series: ARIMA residuals, Prophet anomaly detection

B. Diagnosis Agent

Purpose: Run ML models and rule-based diagnosis

Responsibilities:

- Receive anomaly alerts
- Execute ML model inference
- Apply rule-based refinements
- Generate explainable diagnoses
- Classify urgency (P1/P2/P3)

Rule Engine Examples:

```
# Rule 1: Battery voltage override
if battery_voltage_24h_mean < 12.0 and cranking_voltage < 9.5:
   escalate_risk_level('battery', from_level='MEDIUM', to_level='CRITICAL')
   days_to_failure = 2  # Override model prediction

# Rule 2: Brake pad minimum threshold
if brake_pad_thickness < 2.5:  # Legal minimum in India
   escalate_risk_level('brake_pads', to_level='CRITICAL')
   days_to_failure = 0

# Rule 3: Model-year specific adjustments
if vehicle_model == 'ModelX' and model_year == 2020:
   # Known issue with transmission in 2020 batch
   if component == 'transmission':
      adjust_confidence(multiplier=1.15)
```

Priority Classification:

P1 (Critical - 24-48h):
- Safety-related failures (brakes, steering)
- Imminent breakdown (battery dead, transmission failure)
- Vehicle undrivable

P2 (High - 5-10 days):
- Drivability impact (engine misfires, suspension)
- Preventable breakdown (alternator, fuel pump)
- Expensive cascading failures

P3 (Moderate - 10-30 days):
- Preventive maintenance (filters, fluids)
- Comfort features (AC, power windows)
- Schedule at customer convenience

C. Customer Engagement Agent

Purpose: Proactive voice-first customer communication

Responsibilities:

- Initiate phone calls via Twilio
- Conduct natural conversations using GPT-4
- Handle objections and answer questions
- Provide appointment options
- Fall back to SMS/app if call fails

Technical Specifications:

- **Primary Channel:** Voice (73% conversion rate)
- **Secondary Channel:** App push notification
- **Tertiary Channel:** SMS with booking link
- **Language Support:** English, Hindi, Regional languages
- **Call Duration Target:** <3 minutes average
- **Retry Logic:** 3 attempts over 48 hours

Conversation Architecture:

User Input (Speech)
→ Whisper ASR (Speech-to-Text)
→ Intent Classification (Fine-tuned BERT)
→ Context Management (Conversation state)
→ GPT-4 Response Generation (with custom system prompt)
→ Google Cloud TTS (Text-to-Speech)
→ User Output (Speech)

System Prompt for GPT-4:

You are Maya, a friendly vehicle maintenance assistant for AutoCare.
Your goal is to explain predicted vehicle issues in simple terms and
persuade customers to book preventive maintenance appointments.

Guidelines:
- Be warm, empathetic, and patient
- Use simple language, avoid technical jargon
- Build trust by referencing their vehicle history
- Handle objections with understanding
- Never be pushy; respect customer autonomy

- If unsure, escalate to human advisor

Customer Context:
- Name: {customer_name}
- Vehicle: {vehicle_model} {vehicle_year}
- Last service: {last_service_date}
- Current issue: {predicted_failure}

Start the conversation with a friendly greeting and permission check.

Objection Handling Database: Store common objections with pre-tested response templates:

{
  "objection": "I haven't noticed any problems",
  "response_template": "That's actually quite common! {component} wear happens gradually, so changes aren't noticeable day-to-day. However, our sensors detected {specific_metric} which indicates {explanation}. Think of it like a dental checkup - we catch issues before they become painful and expensive.",
  "effectiveness_score": 0.87
}

D. Scheduling Agent

Purpose: Intelligent appointment optimization

Responsibilities:

- Query service center availability
- Check parts inventory
- Match technician skills to job requirements
- Optimize for customer location and preferences
- Reserve parts upon confirmation
- Send calendar invites and reminders

Optimization Algorithm:

Multi-objective optimization considering:

1. Customer travel distance (minimize)
2. Wait time to appointment (minimize)
3. Parts availability (maximize certainty)
4. Service center load balancing
5. Technician skill match

Scoring Function:

```
def score_appointment_option(center, slot, customer, job):
    score = (
        weight_distance * distance_score(customer.location, center.location) +
        weight_availability * availability_score(slot, job.urgency) +
        weight_parts * parts_availability_score(center, job.parts_needed) +
        weight_skill * technician_match_score(center, job.complexity) +
        weight_balance * load_balance_score(center.current_utilization)
    )
    return score

weights = {
    'distance': 0.30,
    'availability': 0.25,
    'parts': 0.25,
    'skill': 0.15,
    'balance': 0.05
}
```

Parts Reservation System:

- Soft hold (15 minutes) while awaiting customer confirmation
- Hard reservation upon booking
- Auto-release if appointment cancelled
- Substitute part lookup if primary part unavailable

E. Feedback Agent

Purpose: Post-service engagement and data loop closure

Responsibilities:

- Detect service completion
- Send multi-channel feedback requests
- Collect NPS and detailed feedback
- Perform sentiment analysis
- Update vehicle maintenance history
- Trigger quality insights analysis

Feedback Collection Flow:

```
Service Completed
    ↓ (wait 2 hours)
SMS Survey (5-star rating)
```

```
   ↓ (if no response after 4 hours)
App Notification (detailed survey)
   ↓ (if no response after 24 hours)
Voice Call (brief survey)
   ↓
Store feedback + Sentiment Analysis
   ↓
Update vehicle history + Recalibrate predictions
```

Sentiment Analysis:

- Use fine-tuned BERT for automotive service sentiment
- Classify: Positive / Neutral / Negative
- Extract themes: wait_time, cost, staff_behavior, quality, communication
- Flag critical issues (rating ≤2) for immediate escalation

F. Quality Insights Module

Purpose: Manufacturing feedback loop and RCA/CAPA generation

Responsibilities:

- Aggregate field failure patterns
- Correlate with manufacturing data
- Perform statistical root cause analysis
- Generate automated CAPA reports
- Distribute reports to engineering/quality teams
- Track CAPA effectiveness

Failure Pattern Detection:

```
def detect_failure_patterns(time_window_days=30):
    failures = query_service_records(
        service_type='component_failure',
        date_range=last_n_days(time_window_days)
    )

    # Group by component, model, year
    grouped = failures.groupby(['component', 'model_year', 'model'])

    patterns = []
    for (component, year, model), group in grouped:
        failure_rate = len(group) / total_vehicles(year, model)
        baseline_rate = get_historical_failure_rate(component, year, model)
```

```python
    # Statistical significance test
    z_score = (failure_rate - baseline_rate) / std_error(baseline_rate)

    if z_score > 3:  # 3-sigma = statistically significant
        patterns.append({
            'component': component,
            'model_year': year,
            'model': model,
            'failure_count': len(group),
            'failure_rate': failure_rate,
            'expected_rate': baseline_rate,
            'statistical_significance': z_score,
            'affected_vins': list(group['vin']),
            'urgency': classify_urgency(z_score, len(group))
        })

return sorted(patterns, key=lambda x: x['statistical_significance'], reverse=True)
```

Root Cause Analysis Process:

1. **Batch Correlation Analysis:**
    - Query manufacturing DB for affected VINs
    - Check if failures cluster by supplier batch
    - Correlation >70% suggests supplier defect
2. **Time-Based Analysis:**
    - Check if failures from narrow production window
    - Suggests process change or environmental issue
3. **Assembly Line Correlation:**
    - Check if specific line/shift has higher failures
    - Suggests training or equipment issue
4. **Material Testing:**
    - Pull supplier material certificates
    - Check spec compliance
    - Identify out-of-spec parameters
5. **Usage Pattern Analysis:**
    - Compare driving patterns of failed vs non-failed
    - Normal usage + failures = design/manufacturing issue
    - Extreme usage + failures = design margin issue

CAPA Report Generation:

- Automated report using Jinja2 templates
- Include: problem description, RCA, impact assessment, corrective actions, preventive actions
- Distribute via email + Jira ticket creation
- Track implementation status

2.2 Database Design

2.2.1 TimescaleDB (Time-Series Data)

Purpose: Store high-frequency vehicle telemetry

Schema:

```
-- Hypertable for raw telemetry
CREATE TABLE telemetry (
    time TIMESTAMPTZ NOT NULL,
    vin TEXT NOT NULL,
    sensor_type TEXT NOT NULL,
    sensor_value DOUBLE PRECISION,
    unit TEXT,
    quality_flag SMALLINT DEFAULT 0,  -- 0=good, 1=interpolated, 2=suspect
    metadata JSONB
);

SELECT create_hypertable('telemetry', 'time',
    chunk_time_interval => INTERVAL '1 day',
    if_not_exists => TRUE
);

CREATE INDEX idx_telemetry_vin_time ON telemetry (vin, time DESC);
CREATE INDEX idx_telemetry_sensor_type ON telemetry (sensor_type, time DESC);

-- Continuous aggregates for performance
CREATE MATERIALIZED VIEW telemetry_1min
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 minute', time) AS bucket,
    vin,
    sensor_type,
    AVG(sensor_value) AS avg_value,
    MAX(sensor_value) AS max_value,
    MIN(sensor_value) AS min_value,
```

```sql
    STDDEV(sensor_value) AS std_value,
    COUNT(*) AS sample_count
FROM telemetry
GROUP BY bucket, vin, sensor_type;

CREATE MATERIALIZED VIEW telemetry_1hour
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 hour', time) AS bucket,
    vin,
    sensor_type,
    AVG(sensor_value) AS avg_value,
    MAX(sensor_value) AS max_value,
    MIN(sensor_value) AS min_value,
    STDDEV(sensor_value) AS std_value
FROM telemetry
GROUP BY bucket, vin, sensor_type;

-- Retention policy
SELECT add_retention_policy('telemetry', INTERVAL '90 days');
```

Engineered Features Table:

```sql
CREATE TABLE vehicle_features (
    time TIMESTAMPTZ NOT NULL,
    vin TEXT NOT NULL,

    -- Brake features
    brake_pad_thickness_mm DOUBLE PRECISION,
    brake_wear_rate_mm_per_1000km DOUBLE PRECISION,
    hard_brake_count_24h INTEGER,
    brake_pressure_max_1h DOUBLE PRECISION,

    -- Battery features
    battery_voltage_mean_24h DOUBLE PRECISION,
    cranking_voltage DOUBLE PRECISION,
    charge_acceptance_rate DOUBLE PRECISION,
    deep_discharge_events_7d INTEGER,

    -- Engine features
    coolant_temp_max_24h DOUBLE PRECISION,
    oil_pressure_min_7d DOUBLE PRECISION,
    rpm_variance_1h DOUBLE PRECISION,
```

```
    -- Transmission features
    gear_shift_smoothness_score DOUBLE PRECISION,
    transmission_temp_mean_24h DOUBLE PRECISION,

    -- Driving pattern features
    urban_driving_pct DOUBLE PRECISION,
    avg_speed_24h DOUBLE PRECISION,
    aggressive_driving_score DOUBLE PRECISION,

    -- Context features
    ambient_temp_avg DOUBLE PRECISION,
    load_estimate DOUBLE PRECISION,

    PRIMARY KEY (time, vin)
);

SELECT create_hypertable('vehicle_features', 'time');
CREATE INDEX idx_features_vin_time ON vehicle_features (vin, time DESC);
```

2.2.2 PostgreSQL (Relational Data)

Users Table:

```
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email TEXT UNIQUE NOT NULL,
    phone TEXT UNIQUE NOT NULL,
    name TEXT NOT NULL,
    language_preference TEXT DEFAULT 'en',  -- en, hi, ta, te, mr
    timezone TEXT DEFAULT 'Asia/Kolkata',

    -- Communication preferences
    preferred_channel TEXT DEFAULT 'voice',  -- voice, sms, app
    consent_voice_call BOOLEAN DEFAULT TRUE,
    consent_sms BOOLEAN DEFAULT TRUE,
    consent_push BOOLEAN DEFAULT TRUE,
    do_not_disturb_start TIME,
    do_not_disturb_end TIME,

    -- Customer tier
    loyalty_tier TEXT DEFAULT 'regular',  -- regular, silver, gold, platinum

    -- Location
    address_line1 TEXT,
```

```
    address_line2 TEXT,
    city TEXT,
    state TEXT,
    postal_code TEXT,
    country TEXT DEFAULT 'IN',
    gps_location POINT,  -- PostGIS

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),
    last_login_at TIMESTAMPTZ,

    CONSTRAINT valid_channel CHECK (preferred_channel IN ('voice', 'sms', 'app', 'email'))
);

CREATE INDEX idx_users_phone ON users(phone);
CREATE INDEX idx_users_gps ON users USING GIST(gps_location);
```

Vehicles Table:

```
CREATE TABLE vehicles (
    vin TEXT PRIMARY KEY,
    user_id UUID REFERENCES users(id) ON DELETE SET NULL,

    -- Vehicle details
    make TEXT NOT NULL,
    model TEXT NOT NULL,
    variant TEXT,
    model_year INTEGER NOT NULL,
    manufacturing_date DATE,
    production_batch TEXT,
    production_plant TEXT,
    assembly_line TEXT,

    -- Purchase info
    purchase_date DATE,
    purchase_mileage INTEGER DEFAULT 0,

    -- Current status
    current_mileage INTEGER,
    last_telemetry_at TIMESTAMPTZ,
    health_status TEXT DEFAULT 'HEALTHY',  -- HEALTHY, MONITORING, AT_RISK,
CRITICAL

    -- Registration
```

```sql
    registration_number TEXT UNIQUE,
    registration_state TEXT,
    insurance_expiry DATE,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    CONSTRAINT valid_health_status CHECK (health_status IN ('HEALTHY', 'MONITORING',
'AT_RISK', 'CRITICAL'))
);

CREATE INDEX idx_vehicles_user ON vehicles(user_id);
CREATE INDEX idx_vehicles_model ON vehicles(make, model, model_year);
CREATE INDEX idx_vehicles_batch ON vehicles(production_batch);
CREATE INDEX idx_vehicles_health ON vehicles(health_status);
```

Predictions Table:

```sql
CREATE TABLE predictions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    vin TEXT REFERENCES vehicles(vin) ON DELETE CASCADE,
    prediction_time TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Prediction details
    component TEXT NOT NULL,  -- brake_pads, battery, engine, transmission, etc.
    failure_type TEXT,
    risk_level TEXT NOT NULL,  -- NORMAL, LOW, MEDIUM, HIGH, CRITICAL
    confidence DOUBLE PRECISION NOT NULL,  -- 0.0 to 1.0

    -- Timeline
    days_to_failure INTEGER,
    failure_date_estimate DATE,
    confidence_interval_lower INTEGER,
    confidence_interval_upper INTEGER,

    -- Priority
    priority TEXT,  -- P1, P2, P3
    safety_critical BOOLEAN DEFAULT FALSE,
    drivability_impact TEXT,  -- none, minor, moderate, severe

    -- Recommendation
    recommended_action TEXT,
    estimated_service_duration_minutes INTEGER,
    estimated_cost_inr INTEGER,
```

```
    -- Model metadata
    model_name TEXT NOT NULL,
    model_version TEXT NOT NULL,

    -- Explainability
    shap_values JSONB,  -- Top contributing features
    contributing_factors JSONB,  -- Human-readable explanations

    -- Status
    status TEXT DEFAULT 'ACTIVE',  -- ACTIVE, ACKNOWLEDGED, SERVICED,
FALSE_POSITIVE
    acknowledged_at TIMESTAMPTZ,
    serviced_at TIMESTAMPTZ,

    created_at TIMESTAMPTZ DEFAULT NOW(),

    CONSTRAINT valid_risk CHECK (risk_level IN ('NORMAL', 'LOW', 'MEDIUM', 'HIGH',
'CRITICAL')),
    CONSTRAINT valid_priority CHECK (priority IN ('P1', 'P2', 'P3'))
);

CREATE INDEX idx_predictions_vin_time ON predictions(vin, prediction_time DESC);
CREATE INDEX idx_predictions_status ON predictions(status) WHERE status = 'ACTIVE';
CREATE INDEX idx_predictions_component ON predictions(component);
```

Service Centers Table:

```
CREATE TABLE service_centers (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name TEXT NOT NULL,

    -- Location
    address_line1 TEXT NOT NULL,
    address_line2 TEXT,
    city TEXT NOT NULL,
    state TEXT NOT NULL,
    postal_code TEXT NOT NULL,
    gps_location POINT NOT NULL,  -- PostGIS

    -- Contact
    phone TEXT NOT NULL,
    email TEXT,
    manager_name TEXT,
```

```
    -- Capacity
    total_bays INTEGER NOT NULL,
    operating_hours_start TIME DEFAULT '09:00',
    operating_hours_end TIME DEFAULT '18:00',
    weekend_operating BOOLEAN DEFAULT TRUE,

    -- Services offered
    services_offered TEXT[],  -- {brake_service, battery_replacement, engine_repair, ...}

    -- Performance metrics
    average_rating DOUBLE PRECISION DEFAULT 0,
    total_reviews INTEGER DEFAULT 0,
    current_utilization_pct DOUBLE PRECISION DEFAULT 0,

    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_centers_gps ON service_centers USING GIST(gps_location);
CREATE INDEX idx_centers_city ON service_centers(city, state);
```

Parts Inventory Table:

```
CREATE TABLE parts_inventory (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_center_id UUID REFERENCES service_centers(id) ON DELETE CASCADE,

    -- Part details
    part_number TEXT NOT NULL,
    part_name TEXT NOT NULL,
    part_category TEXT,  -- brake, battery, engine, transmission, electrical
    compatible_models TEXT[],  -- {ModelA, ModelB, ...}

    -- Inventory
    quantity_available INTEGER NOT NULL DEFAULT 0,
    quantity_reserved INTEGER NOT NULL DEFAULT 0,
    minimum_stock_level INTEGER DEFAULT 5,

    -- Supplier
    supplier_name TEXT,
    supplier_batch TEXT,
    unit_cost_inr DOUBLE PRECISION,
```

```
    -- Reorder
    reorder_point INTEGER,
    reorder_quantity INTEGER,
    lead_time_days INTEGER,

    last_restocked_at TIMESTAMPTZ,
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(service_center_id, part_number)
);

CREATE INDEX idx_inventory_center_part ON parts_inventory(service_center_id,
part_number);
CREATE INDEX idx_inventory_low_stock ON parts_inventory(service_center_id)
    WHERE quantity_available <= minimum_stock_level;
```

Appointments Table:

```
CREATE TABLE appointments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    vin TEXT REFERENCES vehicles(vin) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    service_center_id UUID REFERENCES service_centers(id),
    prediction_id UUID REFERENCES predictions(id),

    -- Scheduling
    appointment_time TIMESTAMPTZ NOT NULL,
    estimated_duration_minutes INTEGER NOT NULL,
    appointment_end_time TIMESTAMPTZ GENERATED ALWAYS AS
        (appointment_time + (estimated_duration_minutes || ' minutes')::INTERVAL) STORED,

    -- Service details
    service_type TEXT NOT NULL,
    components_to_service TEXT[],
    parts_required JSONB,  -- [{part_number, quantity, reserved}]
    assigned_technician_id UUID,
    estimated_cost_inr INTEGER,

    -- Status tracking
    status TEXT DEFAULT 'SCHEDULED',  -- SCHEDULED, CONFIRMED, IN_PROGRESS,
COMPLETED, CANCELLED, NO_SHOW
    booking_source TEXT,  -- voice, app, sms, web
    booking_agent_id TEXT,  -- Which agent booked this
```

```sql
    -- Communication tracking
    confirmation_sent_at TIMESTAMPTZ,
    reminder_sent_at TIMESTAMPTZ,
    customer_confirmed_at TIMESTAMPTZ,

    -- Completion
    checked_in_at TIMESTAMPTZ,
    service_started_at TIMESTAMPTZ,
    service_completed_at TIMESTAMPTZ,
    actual_cost_inr INTEGER,

    -- Cancellation
    cancelled_at TIMESTAMPTZ,
    cancellation_reason TEXT,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW(),

    CONSTRAINT valid_status CHECK (status IN ('SCHEDULED', 'CONFIRMED',
'IN_PROGRESS', 'COMPLETED', 'CANCELLED', 'NO_SHOW'))
);

CREATE INDEX idx_appointments_vin ON appointments(vin);
CREATE INDEX idx_appointments_user ON appointments(user_id);
CREATE INDEX idx_appointments_center_time ON appointments(service_center_id,
appointment_time);
CREATE INDEX idx_appointments_status ON appointments(status) WHERE status IN
('SCHEDULED', 'CONFIRMED');
```

Maintenance History Table:

```sql
CREATE TABLE maintenance_history (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    vin TEXT REFERENCES vehicles(vin) ON DELETE CASCADE,
    appointment_id UUID REFERENCES appointments(id),

    -- Service details
    service_date DATE NOT NULL,
    service_type TEXT NOT NULL,
    mileage_at_service INTEGER,

    -- Work performed
    components_serviced TEXT[],
```

```
    parts_replaced JSONB,  -- [{part_number, part_name, quantity, cost}]
    labor_hours DOUBLE PRECISION,

    -- Cost breakdown
    parts_cost_inr INTEGER,
    labor_cost_inr INTEGER,
    total_cost_inr INTEGER,

    -- Technician
    technician_id UUID,
    technician_name TEXT,
    technician_notes TEXT,

    -- Quality
    service_center_id UUID REFERENCES service_centers(id),
    warranty_applicable BOOLEAN DEFAULT TRUE,
    warranty_months INTEGER DEFAULT 6,

    -- RCA tags (for quality analysis)
    failure_mode TEXT,
    root_cause_category TEXT,
    capa_id TEXT,  -- Link to CAPA report if generated

    created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_maintenance_vin_date ON maintenance_history(vin, service_date DESC);
CREATE INDEX idx_maintenance_components ON maintenance_history USING
GIN(components_serviced);
CREATE INDEX idx_maintenance_capa ON maintenance_history(capa_id) WHERE capa_id IS
NOT NULL;
```

Feedback Table:

```
CREATE TABLE feedback (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    appointment_id UUID REFERENCES appointments(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id),
    vin TEXT REFERENCES vehicles(vin),

    -- Ratings (1-5 scale)
    overall_rating SMALLINT CHECK (overall_rating BETWEEN 1 AND 5),
    service_quality_rating SMALLINT CHECK (service_quality_rating BETWEEN 1 AND 5),
    staff_behavior_rating SMALLINT CHECK (staff_behavior_rating BETWEEN 1 AND 5),
```

```
    wait_time_rating SMALLINT CHECK (wait_time_rating BETWEEN 1 AND 5),
    facility_cleanliness_rating SMALLINT CHECK (facility_cleanliness_rating BETWEEN 1 AND 5),
    value_for_money_rating SMALLINT CHECK (value_for_money_rating BETWEEN 1 AND 5),

    -- NPS
    nps_score SMALLINT CHECK (nps_score BETWEEN 0 AND 10),
    nps_category TEXT GENERATED ALWAYS AS (
        CASE
            WHEN nps_score >= 9 THEN 'PROMOTER'
            WHEN nps_score >= 7 THEN 'PASSIVE'
            ELSE 'DETRACTOR'
        END
    ) STORED,

    -- Detailed feedback
    comments TEXT,
    positive_aspects TEXT[],
    improvement_areas TEXT[],

    -- Sentiment analysis
    sentiment TEXT,  -- POSITIVE, NEUTRAL, NEGATIVE
    sentiment_score DOUBLE PRECISION,  -- -1.0 to 1.0
    themes_extracted TEXT[],  -- [wait_time, cost, staff_behavior, ...]

    -- Collection metadata
    collection_channel TEXT,  -- sms, app, voice
    collected_at TIMESTAMPTZ DEFAULT NOW(),

    -- Follow-up
    requires_follow_up BOOLEAN DEFAULT FALSE,
    follow_up_completed BOOLEAN DEFAULT FALSE,
    follow_up_notes TEXT,

    created_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_feedback_appointment ON feedback(appointment_id);
CREATE INDEX idx_feedback_nps ON feedback(nps_category);
CREATE INDEX idx_feedback_low_rating ON feedback(overall_rating) WHERE overall_rating <= 3;
```

Agent Audit Log:

```sql
CREATE TABLE agent_audit_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

    -- Agent identification
    agent_name TEXT NOT NULL,
    agent_type TEXT NOT NULL,  -- master, data_analysis, diagnosis, engagement, scheduling,
feedback, quality
    agent_instance_id TEXT,  -- For distributed deployments

    -- Action details
    action_type TEXT NOT NULL,  -- API_CALL, DATABASE_QUERY, MESSAGE_SENT,
PREDICTION_MADE, etc.
    action_description TEXT,

    -- Context
    vehicle_id TEXT,
    user_id UUID,
    appointment_id UUID,

    -- Request/Response
    request_payload JSONB,
    response_payload JSONB,

    -- Result
    status TEXT NOT NULL,  -- SUCCESS, FAILURE, BLOCKED, TIMEOUT
    error_message TEXT,

    -- Performance
    execution_time_ms INTEGER,

    -- Security (UEBA)
    anomaly_score DOUBLE PRECISION,
    anomaly_flags TEXT[],  -- [UNUSUAL_TIME, UNAUTHORIZED_TABLE,
HIGH_FREQUENCY, ...]
    ueba_action_taken TEXT,  -- ALLOWED, BLOCKED, THROTTLED, ALERTED

    -- Timestamps
    timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Retention
    partition_key TEXT GENERATED ALWAYS AS (to_char(timestamp, 'YYYY-MM')) STORED
);

CREATE INDEX idx_audit_agent_time ON agent_audit_log(agent_name, timestamp DESC);
```

```sql
CREATE INDEX idx_audit_status ON agent_audit_log(status) WHERE status IN ('FAILURE',
'BLOCKED');
CREATE INDEX idx_audit_anomaly ON agent_audit_log(anomaly_score DESC) WHERE
anomaly_score >= 75;

-- Partition by month for performance
CREATE TABLE agent_audit_log_2025_10 PARTITION OF agent_audit_log
  FOR VALUES IN ('2025-10');
```

Manufacturing Data Integration Tables:

```sql
-- Production records (synced from manufacturing ERP)
CREATE TABLE production_records (
  vin TEXT PRIMARY KEY,
  production_date DATE NOT NULL,
  production_plant TEXT NOT NULL,
  assembly_line TEXT NOT NULL,
  shift TEXT,  -- morning, afternoon, night

  -- Quality control
  qc_passed BOOLEAN DEFAULT TRUE,
  qc_test_results JSONB,
  qc_inspector_id TEXT,

  -- Supplier batches (critical for RCA)
  engine_batch TEXT,
  transmission_batch TEXT,
  brake_system_batch TEXT,
  battery_batch TEXT,
  electrical_system_batch TEXT,

  -- Environmental conditions during production
  ambient_temp_celsius DOUBLE PRECISION,
  humidity_pct DOUBLE PRECISION,

  created_at TIMESTAMPTZ DEFAULT NOW(),
  imported_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_production_date ON production_records(production_date);
CREATE INDEX idx_production_plant_line ON production_records(production_plant,
assembly_line);
CREATE INDEX idx_production_brake_batch ON production_records(brake_system_batch);
```

```sql
-- CAPA records
CREATE TABLE capa_records (
   id TEXT PRIMARY KEY,  -- e.g., CAPA-2024-Q4-0047

   -- Problem identification
   component TEXT NOT NULL,
   failure_mode TEXT NOT NULL,
   severity_level TEXT NOT NULL,  -- LOW, MEDIUM, HIGH, CRITICAL

   -- Affected vehicles
   affected_vins TEXT[],
   affected_count INTEGER,
   vehicles_at_risk INTEGER,

   -- Root cause
   root_cause_hypothesis TEXT,
   root_cause_confidence TEXT,  -- LOW, MEDIUM, HIGH
   supplier_batch TEXT,
   production_window_start DATE,
   production_window_end DATE,

   -- Impact assessment
   customer_impact TEXT,
   financial_impact_inr INTEGER,
   regulatory_impact TEXT,

   -- Actions
   corrective_actions JSONB,  -- [{action, responsible, deadline, status}]
   preventive_actions JSONB,

   -- Status tracking
   status TEXT DEFAULT 'OPEN',  -- OPEN, IN_PROGRESS, VERIFICATION, CLOSED
   opened_date DATE NOT NULL,
   target_closure_date DATE,
   actual_closure_date DATE,

   -- Approvals
   prepared_by TEXT,
   reviewed_by TEXT,
   approved_by TEXT[],

   -- Effectiveness
   effectiveness_measured BOOLEAN DEFAULT FALSE,
   effectiveness_score DOUBLE PRECISION,  -- 0-100
```

```
  failure_rate_before DOUBLE PRECISION,
  failure_rate_after DOUBLE PRECISION,

  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_capa_status ON capa_records(status);
CREATE INDEX idx_capa_severity ON capa_records(severity_level);
CREATE INDEX idx_capa_component ON capa_records(component);
```

2.3 API Design

2.3.1 Public APIs (External Integrations)

Base URL: `https://api.autocare.com/v1`

Authentication:

- API Key (for vehicle/telemetry ingestion)
- JWT Bearer Token (for user-facing APIs)
- OAuth 2.0 (for third-party integrations)

Rate Limiting:

- Telemetry ingestion: 10,000 requests/minute per API key
- User APIs: 100 requests/minute per user
- Admin APIs: 1,000 requests/minute per account

Telemetry Ingestion:

```
POST /telemetry/ingest
Content-Type: application/json
X-API-Key: {api_key}

Request Body:
{
  "vin": "1HGBH41JXMN109186",
  "timestamp": "2025-10-13T10:30:00Z",
  "sensors": [
    {"type": "speed", "value": 65.5, "unit": "km/h"},
    {"type": "rpm", "value": 2800, "unit": "rpm"},
    {"type": "battery_voltage", "value": 12.6, "unit": "V"},
    {"type": "brake_pad_thickness", "value": 4.2, "unit": "mm"},
```

```
    {"type": "coolant_temp", "value": 92, "unit": "celsius"},
    {"type": "oil_pressure", "value": 35, "unit": "psi"}
  ],
  "gps": {
    "latitude": 12.9716,
    "longitude": 77.5946,
    "altitude": 920
  },
  "odometer": 18420,
  "fuel_level": 45.5,
  "dtc_codes": []
}
```

Response (200 OK):

```
{
  "status": "accepted",
  "message_id": "msg_abc123",
  "processed_at": "2025-10-13T10:30:01Z"
}
```

Batch Telemetry Ingestion:

```
POST /telemetry/batch
Content-Type: application/json
X-API-Key: {api_key}
```

Request Body:

```
{
  "vin": "1HGBH41JXMN109186",
  "readings": [
    {
      "timestamp": "2025-10-13T10:30:00Z",
      "sensors": [...]
    },
    {
      "timestamp": "2025-10-13T10:31:00Z",
      "sensors": [...]
    }
  ]
}
```

Vehicle Registration:

```
POST /vehicles
Authorization: Bearer {jwt_token}

Request Body:
{
  "vin": "1HGBH41JXMN109186",
  "make": "Honda",
  "model": "City",
  "variant": "VX",
  "model_year": 2023,
  "purchase_date": "2023-03-15",
  "purchase_mileage": 0,
  "registration_number": "KA01AB1234"
}

Response (201 Created):
{
  "vin": "1HGBH41JXMN109186",
  "status": "registered",
  "health_monitoring_enabled": true,
  "telemetry_api_key": "veh_key_xyz789"
}
```

Vehicle Health Status:

```
GET /vehicles/{vin}/health
Authorization: Bearer {jwt_token}

Response (200 OK):
{
  "vin": "1HGBH41JXMN109186",
  "health_status": "AT_RISK",
  "last_updated": "2025-10-13T10:30:00Z",
  "current_mileage": 18420,
  "predictions": [
    {
      "component": "brake_pads",
      "risk_level": "HIGH",
      "confidence": 0.89,
      "days_to_failure": 12,
      "estimated_cost": 2800,
      "recommended_action": "Schedule brake pad replacement within 7 days"
    }
  ],
```

```
  "upcoming_appointments": [
    {
      "appointment_id": "appt_123",
      "service_center": "Indiranagar Service Center",
      "appointment_time": "2025-10-20T10:00:00Z"
    }
  ]
}
```

Appointment Booking:

POST /appointments
Authorization: Bearer {jwt_token}

Request Body:

```
{
  "vin": "1HGBH41JXMN109186",
  "service_type": "brake_pad_replacement",
  "preferred_date": "2025-10-20",
  "preferred_time_slot": "morning",  // morning, afternoon, evening
  "notes": "Customer requested weekend appointment"
}
```

Response (201 Created):

```
{
  "appointment_id": "appt_123",

 "appointment_id": "appt_123",
  "vin": "1HGBH41JXMN109186",
  "service_center": {
    "id": "center_001",
    "name": "Indiranagar Service Center",
    "address": "123 Main Road, Indiranagar, Bangalore 560038",
    "phone": "+91-80-12345678",
    "distance_km": 4.2
  },
  "appointment_time": "2025-10-20T10:00:00Z",
  "estimated_duration_minutes": 90,
  "estimated_cost": 2800,
  "parts_reserved": true,
  "status": "CONFIRMED",
  "calendar_invite_sent": true
}
```

Trip Planning / Pre-Trip Check:

POST /vehicles/{vin}/trip-check
Authorization: Bearer {jwt_token}

Request Body:


```
{
  "origin": {
    "latitude": 12.9716,
    "longitude": 77.5946
  },
  "destination": {
    "latitude": 15.2993,
    "longitude": 74.1240
  },
  "departure_date": "2025-10-25",
  "estimated_duration_hours": 8,
  "passenger_count": 4
}
```


Response (200 OK):


```
{
  "trip_safe": false,
  "recommendations": [
    {
      "priority": "HIGH",
      "component": "brake_pads",
      "issue": "Current brake pad thickness is 2.8mm. For a 500km highway trip with 4 passengers, we recommend replacement to ensure safety.",
      "estimated_wear_during_trip": "0.6mm",
      "risk_level": "HIGH",
      "action_required": "Replace brake pads before trip"
    }
  ],
  "suggested_services": [
    {
      "service": "brake_pad_replacement",
      "urgency": "before_trip",
      "estimated_cost": 2800
    },
    {
      "service": "tire_pressure_check",
      "urgency": "recommended",
      "estimated_cost": 0
```

    }
  ],
  "nearby_service_centers": [
    {
      "id": "center_001",
      "name": "Indiranagar Service Center",
      "distance_km": 4.2,
      "available_slots": ["2025-10-23T09:00:00Z", "2025-10-23T14:00:00Z"]
    }
  ]
}

Feedback Submission:

POST /appointments/{appointment_id}/feedback
Authorization: Bearer {jwt_token}

Request Body:
{
  "overall_rating": 5,
  "service_quality_rating": 5,
  "staff_behavior_rating": 5,
  "wait_time_rating": 4,
  "facility_cleanliness_rating": 5,
  "nps_score": 9,
  "comments": "Excellent service! The technician explained everything clearly.",
  "would_recommend": true
}

Response (201 Created):
{
  "feedback_id": "fb_456",
  "status": "received",
  "thank_you_points_awarded": 100
}

2.3.2 Internal APIs (Agent Communication)

Master Agent Orchestration:

POST /internal/agents/task
X-Agent-Auth: {internal_token}

Request Body:
{
  "task_id": "task_abc123",
  "task_type": "CUSTOMER_ENGAGEMENT",
  "priority": "HIGH",
  "payload": {
    "vehicle_id": "1HGBH41JXMN109186",
    "prediction": {
      "component": "brake_pads",
      "risk_level": "HIGH",
      "confidence": 0.89,
      "days_to_failure": 12
    },
    "customer_profile": {
      "name": "Rajesh Sharma",
      "phone": "+91-9876543210",
      "language": "en",
      "preferred_channel": "voice"
    }
  },
  "timeout_seconds": 300
}

Response (202 Accepted):
{
  "task_id": "task_abc123",
  "status": "QUEUED",
  "estimated_start_time": "2025-10-13T10:35:00Z"
}

Model Inference:

POST /internal/models/predict
X-Agent-Auth: {internal_token}

Request Body:
{
  "vin": "1HGBH41JXMN109186",
  "features": {
    "brake_pad_thickness_mm": 3.2,
    "brake_wear_rate_mm_per_1000km": 0.8,
    "hard_brake_count_24h": 15,
    "urban_driving_pct": 75,
    "vehicle_age_months": 18,

    "avg_speed_24h": 32.5
  },
  "model_name": "brake_failure_predictor",
  "explain": true
}

Response (200 OK):

{
  "prediction": {
    "failure_probability": 0.89,
    "risk_class": "HIGH",
    "days_to_failure": 12,
    "confidence_interval": [9, 15]
  },
  "explanation": {
    "shap_values": [
      {"feature": "brake_wear_rate_mm_per_1000km", "contribution": 0.34},
      {"feature": "brake_pad_thickness_mm", "contribution": 0.28},
      {"feature": "hard_brake_count_24h", "contribution": 0.19}
    ],
    "human_readable": "High wear rate (0.8mm/1000km) and frequent hard braking are the primary risk factors."
  },
  "model_version": "v2.3.1",
  "inference_time_ms": 42
}

UEBA Check:

POST /internal/ueba/validate
X-Agent-Auth: {internal_token}

Request Body:

{
  "agent_name": "scheduling_agent_01",
  "action_type": "DATABASE_QUERY",
  "action_details": {
    "query_type": "SELECT",
    "table": "service_centers",
    "row_count": 3,
    "payload_size_bytes": 1024
  },
  "context": {
    "timestamp": "2025-10-13T10:30:00Z",

"source_ip": "10.0.1.45"
}
}

Response (200 OK):

{
"allowed": true,
"anomaly_score": 12,
"risk_level": "LOW",
"action": "ALLOWED",
"baseline_deviation": 0.3
}

2.4 Message Queue Architecture

Kafka Topics:

```yaml
topics:
  # Raw telemetry ingestion
  - name: vehicle.telemetry.raw
    partitions: 50
    replication_factor: 3
    retention_ms: 604800000  # 7 days

  # Processed features
  - name: vehicle.features.computed
    partitions: 50
    replication_factor: 3
    retention_ms: 2592000000  # 30 days

  # Anomaly alerts
  - name: vehicle.anomalies.detected
    partitions: 20
    replication_factor: 3
    retention_ms: 2592000000  # 30 days

  # Predictions
  - name: vehicle.predictions.generated
    partitions: 20
    replication_factor: 3
    retention_ms: 7776000000  # 90 days

  # Agent tasks (RabbitMQ handles better)
```

```
  # - customer_engagement_tasks
  # - scheduling_tasks
  # - feedback_tasks
```

RabbitMQ Queues:

```yaml
queues:
  # Worker agent task queues
  - name: tasks.customer_engagement
    durable: true
    message_ttl: 3600000  # 1 hour
    max_priority: 10
    dead_letter_exchange: dlx.tasks

  - name: tasks.scheduling
    durable: true
    message_ttl: 1800000  # 30 minutes
    max_priority: 10
    dead_letter_exchange: dlx.tasks

  - name: tasks.feedback
    durable: true
    message_ttl: 86400000  # 24 hours
    max_priority: 5
    dead_letter_exchange: dlx.tasks

  - name: tasks.quality_insights
    durable: true
    message_ttl: 604800000  # 7 days
    max_priority: 5
    dead_letter_exchange: dlx.tasks

  # Dead letter queues
  - name: dlx.tasks.customer_engagement
    durable: true

  # Priority routing
  exchanges:
    - name: ex.tasks
      type: topic
      durable: true
      routing_keys:
        - "task.engagement.#"
        - "task.scheduling.#"
```

```
    - "task.feedback.#"
```

2.5 Caching Strategy

Redis Cache Layers:

```
cache_layers:
  # Hot vehicle state (very frequently accessed)
  - name: vehicle_state
    prefix: "veh:{vin}:state"
    ttl: 3600  # 1 hour
    eviction: LRU
    example_key: "veh:1HGBH41JXMN109186:state"
    example_value:
      last_telemetry: "2025-10-13T10:30:00Z"
      health_status: "AT_RISK"
      active_predictions: [{component: "brake_pads", confidence: 0.89}]
      scheduled_appointment: "2025-10-20T10:00:00Z"

  # Recent predictions
  - name: predictions_cache
    prefix: "pred:{vin}:{component}"
    ttl: 7200  # 2 hours
    eviction: LRU

  # Customer profiles
  - name: customer_profiles
    prefix: "user:{user_id}:profile"
    ttl: 1800  # 30 minutes
    eviction: LRU

  # Service center availability
  - name: center_availability
    prefix: "center:{center_id}:slots:{date}"
    ttl: 300  # 5 minutes (frequent updates)
    eviction: LRU

  # Parts inventory
  - name: parts_inventory
    prefix: "inventory:{center_id}:{part_number}"
    ttl: 600  # 10 minutes
    eviction: LRU
```

```yaml
  # Model inference results (short-lived)
  - name: inference_cache
    prefix: "inference:{model}:{feature_hash}"
    ttl: 60  # 1 minute
    eviction: LFU

  # Rate limiting
  - name: rate_limits
    prefix: "ratelimit:{endpoint}:{api_key}"
    ttl: 60  # 1 minute sliding window
    eviction: None (don't evict)
```

Cache Invalidation Patterns:

1. **Write-Through:** Updates written to DB first, then cache updated
2. **Cache-Aside:** Read from cache, if miss, read from DB and populate cache
3. **Event-Based:** Kafka events trigger cache invalidation
4. **TTL-Based:** Natural expiration for less critical data

2.6 Security Implementation

2.6.1 UEBA (User and Entity Behavior Analytics)

Baseline Learning Process (30-day training):

For each agent, collect:

- API call patterns (frequency, endpoints, payload sizes)
- Database access patterns (tables, query types, row counts)
- Execution patterns (duration, success rate, error types)
- Communication patterns (message queue activity)
- Temporal patterns (time-of-day, day-of-week)

Anomaly Detection Algorithm:

```python
def calculate_anomaly_score(agent_id, current_behavior):
    baseline = get_agent_baseline(agent_id)
    score = 0
    violations = []

    # 1. API Call Anomalies (30%)
    for api, calls in current_behavior.api_calls.items():
        if api not in baseline.allowed_apis:
            score += 40
```

```
            violations.append(f"UNAUTHORIZED_API: {api}")
        elif calls.frequency > baseline.mean + (3 * baseline.std):
            score += 15
            violations.append(f"HIGH_FREQUENCY: {api}")

    # 2. Data Access Anomalies (30%)
    for table in current_behavior.tables_accessed:
        if table in baseline.forbidden_tables:
            score += 50
            violations.append(f"FORBIDDEN_TABLE: {table}")
        if current_behavior.row_count[table] > baseline.typical_rows * 5:
            score += 20
            violations.append(f"EXCESSIVE_DATA_ACCESS: {table}")

    # 3. Temporal Anomalies (15%)
    if is_unusual_time(current_behavior.timestamp, baseline.activity_hours):
        score += 10
        violations.append("UNUSUAL_TIME")

    # 4. Execution Anomalies (15%)
    if current_behavior.duration > baseline.mean_duration + (4 * baseline.std_duration):
        score += 10
        violations.append("SLOW_EXECUTION")
    if current_behavior.error_rate > baseline.error_rate * 3:
        score += 10
        violations.append("HIGH_ERROR_RATE")

    # 5. Communication Anomalies (10%)
    if current_behavior.message_count > baseline.mean_messages + (5 *
baseline.std_messages):
        score += 15
        violations.append("EXCESSIVE_MESSAGING")

    return {
        'anomaly_score': min(score, 100),
        'violations': violations,
        'severity': classify_severity(score)
    }
```

Automated Response Actions:

```
def handle_anomaly(anomaly_result):
    score = anomaly_result['anomaly_score']
```

```
if score >= 90:  # CRITICAL
    disable_agent(agent_id)
    revoke_api_credentials(agent_id)
    send_pagerduty_alert()
    preserve_forensic_evidence(agent_id)

elif score >= 75:  # HIGH
    block_current_action(agent_id)
    send_security_alert()
    require_manual_review()

elif score >= 50:  # MEDIUM
    log_warning()
    increase_monitoring_frequency(from_5min_to_1min)
    send_ops_notification()

else:  # LOW
    log_info()
```

2.6.2 Data Encryption

At Rest:

- AES-256 encryption for all databases
- Encrypted EBS volumes for Kubernetes persistent volumes
- S3 server-side encryption (SSE-S3 or SSE-KMS)
- Encrypted Redis with AUTH

In Transit:

- TLS 1.3 for all API communications
- mTLS for inter-service communication
- WSS (WebSocket Secure) for real-time updates
- MQTT over TLS for vehicle telemetry

PII Protection:

- Tokenization for phone numbers and emails
- Hash VINs in logs
- Field-level encryption for sensitive customer data
- Data masking in non-production environments

2.6.3 Authentication & Authorization

User Authentication:

- JWT tokens with 1-hour expiration
- Refresh tokens with 30-day expiration
- OAuth 2.0 for third-party integrations
- MFA for admin users

Service-to-Service:

- mTLS certificates
- Service accounts with scoped permissions
- API keys with IP whitelisting
- Internal token validation via shared secret

RBAC Roles:

```yaml
roles:
  - name: customer
    permissions:
      - read:own_vehicles
      - write:own_appointments
      - write:own_feedback

  - name: service_advisor
    permissions:
      - read:assigned_appointments
      - write:service_records
      - read:vehicle_diagnostics

  - name: service_center_manager
    permissions:
      - read:center_appointments
      - write:center_inventory
      - read:center_analytics

  - name: engineer
    permissions:
      - read:all_predictions
      - read:manufacturing_data
      - write:capa_records

  - name: admin
    permissions:
      - "*:*"
```

---

3. FRONTEND ARCHITECTURE

3.1 Technology Stack

Core Framework:

- Next.js 14 (App Router) with React Server Components
- React 18 with TypeScript
- Server-side rendering for SEO and performance

3D Visualization:

- React Three Fiber (R3F) for 3D rendering
- Three.js r160+ as the underlying engine
- @react-three/drei for helper components
- @react-three/postprocessing for effects (bloom, DOF)

UI Components:

- Tailwind CSS for styling
- shadcn/ui for component library
- Framer Motion for animations
- Lucide React for icons

Data Visualization:

- ECharts for complex charts
- Recharts for simple charts
- D3.js for custom visualizations

State Management:

- Zustand for global state
- React Query (TanStack Query) for server state
- Context API for theme/auth

Real-Time Communication:

- Socket.io for live updates
- Server-Sent Events for notifications

3.2 Site Structure & Pages

3.2.1 Page Architecture

/

```
├── (public)
│   ├── page.tsx              # Landing page with hero
│   ├── explore/
│   │   └── page.tsx          # Interactive 3D car explorer
│   ├── about/
│   │   └── page.tsx          # About the platform
│   ├── contact/
│   │   └── page.tsx          # Contact form
│   └── login/
│       └── page.tsx          # Authentication
│
├── (authenticated)
│   ├── dashboard/
│   │   └── page.tsx          # Main dashboard / vehicle list
│   ├── vehicles/
│   │   └── [vin]/
│   │       ├── page.tsx      # Vehicle detail with 3D model
│   │       ├── history/
│   │       │   └── page.tsx  # Maintenance history
│   │       └── trip-planner/
│   │           └── page.tsx  # Pre-trip safety check
│   ├── appointments/
│   │   ├── page.tsx          # Appointment list
│   │   ├── book/
│   │   │   └── page.tsx      # Book new appointment
│   │   └── [id]/
│   │       └── page.tsx      # Appointment detail
│   ├── insights/
│   │   └── page.tsx          # Fleet insights (multi-vehicle)
│   └── profile/
│       └── page.tsx          # User settings
│
└── (admin)
    ├── admin/
    │   ├── dashboard/
    │   │   └── page.tsx      # Admin overview
    │   ├── agents/
    │   │   └── page.tsx      # Agent monitoring (UEBA)
    │   ├── service-centers/
    │   │   └── page.tsx      # Service center management
    │   └── quality/
    │       ├── page.tsx      # CAPA dashboard
    │       └── [id]/
    │           └── page.tsx  # CAPA detail
    │
```

3.3 Key Components & Features

3.3.1 Landing Page

Hero Section with 3D Car:

```
// components/Hero3D.tsx
'use client'

import { Canvas } from '@react-three/fiber'
import { OrbitControls, Environment, useGLTF } from '@react-three/drei'
import { Suspense } from 'react'

export function Hero3D() {
  return (
    <div className="h-screen w-full">
      <Canvas camera={{ position: [5, 2, 5], fov: 50 }}>
        <Suspense fallback={<LoadingSpinner />}>
          <Environment preset="sunset" />
          <CarModel />
          <OrbitControls
            enableZoom={false}
            enablePan={false}
            maxPolarAngle={Math.PI / 2}
            minPolarAngle={Math.PI / 3}
          />
        </Suspense>
      </Canvas>

      <div className="absolute inset-0 flex items-center justify-center pointer-events-none">
        <div className="text-center pointer-events-auto">
          <h1 className="text-6xl font-bold mb-4">
            Never Break Down Again
          </h1>
          <p className="text-xl mb-8">
            AI-powered predictive maintenance for your vehicle
          </p>
          <button className="px-8 py-3 bg-blue-600 rounded-lg">
            Get Started
          </button>
        </div>
      </div>
    </div>
```

```
  )
}
```

Value Proposition Cards:

- Real-time monitoring with live sensor visualization
- 93% reduction in breakdowns (animated counter)
- Autonomous scheduling (chatbot demo)
- Manufacturing quality feedback (infographic)

Social Proof:

- Customer testimonials with photos
- Service center partner logos
- Statistics with animated counters

3.3.2 Vehicle Detail Page (Core UX)

Layout:

```
┌─────────────────────────────────────────────────────────┐
│ Header: Vehicle Name, Registration, Mileage       │
├─────────────────────┬───────────────────────────────────┤
│            │ Health Summary Card          │
│            │ ┌─────────────────────────────┐ │
│ Interactive    │ │ Status: AT RISK        │ │
│ 3D Car Model    │ │ Confidence: 89%        │ │
│ (Click parts)   │ │ Next Service: 12 days      │ │
│            │ └─────────────────────────────┘ │
│            │                │
│            │ Component Health Cards       │
│            │ ┌──────┐ ┌──────┐ ┌──────┐   │
│            │ │Brake│ │Batt │ │Eng │    │
│            │ │ 89% │ │ 95% │ │ 92% │    │
│            │ └──────┘ └──────┘ └──────┘   │
├─────────────────────┴───────────────────────────────────┤
│ Timeline Scrubber (Historical Data)          │
│ ──────●───────────────────────────────────────────────────        │
│ Oct 1  Oct 5  Oct 10  Oct 13 (Today)         │
├─────────────────────────────────────────────────────────┤
│ Detailed Charts                   │
│ [Brake Pad Thickness] [Battery Voltage] [RPM]     │
└─────────────────────────────────────────────────────────┘
```

3D Car Interaction:

```
// components/InteractiveCar.tsx
'use client'

import { Canvas } from '@react-three/fiber'
import { useGLTF, Html } from '@react-three/drei'
import { useState } from 'react'

function CarModel({ onPartClick, highlightedPart }) {
  const { scene, nodes } = useGLTF('/models/car.glb')

  const parts = {
    brake_front_left: nodes.BrakeFrontLeft,
    brake_front_right: nodes.BrakeFrontRight,
    battery: nodes.Battery,
    engine: nodes.Engine,
    transmission: nodes.Transmission
  }

  return (
    <group>
      {Object.entries(parts).map(([partName, mesh]) => (
        <mesh
          key={partName}
          geometry={mesh.geometry}
          material={mesh.material.clone()}
          onClick={() => onPartClick(partName)}
          onPointerOver={(e) => {
            e.stopPropagation()
            document.body.style.cursor = 'pointer'
            // Glow effect
            e.object.material.emissive.set(0x4444ff)
            e.object.material.emissiveIntensity = 0.3
          }}
          onPointerOut={(e) => {
            document.body.style.cursor = 'default'
            e.object.material.emissive.set(0x000000)
          }}
        >
          {highlightedPart === partName && (
            <Html position={[0, 0.5, 0]}>
              <div className="bg-black/80 text-white px-3 py-2 rounded">
                Click for details
```

```
          </div>
        </Html>
      )}
    </mesh>
  ))}
</group>
)
}

export function InteractiveCar({ vehicleData }) {
  const [selectedPart, setSelectedPart] = useState(null)

  return (
    <div className="relative h-full">
      <Canvas>
        <CarModel
          onPartClick={setSelectedPart}
          highlightedPart={selectedPart}
        />
      </Canvas>

      {selectedPart && (
        <PartDetailPanel
          part={selectedPart}
          data={vehicleData.predictions[selectedPart]}
          onClose={() => setSelectedPart(null)}
        />
      )}
    </div>
  )
}
```

Component Health Cards:

```
// components/ComponentHealthCard.tsx
interface ComponentHealth {
  component: string
  health_score: number  // 0-100
  status: 'healthy' | 'warning' | 'critical'
  last_service: string
  predicted_failure_days: number | null
}

export function ComponentHealthCard({ component }: { component: ComponentHealth }) {
```

```
  const statusColors = {
    healthy: 'bg-green-500',
    warning: 'bg-yellow-500',
    critical: 'bg-red-500'
  }

  return (
    <div className="border rounded-lg p-4 hover:shadow-lg transition">
      <div className="flex items-center justify-between mb-3">
        <h3 className="font-semibold">{component.component}</h3>
        <span className={`h-3 w-3 rounded-full ${statusColors[component.status]}`} />
      </div>

      <div className="mb-3">
        <div className="flex justify-between text-sm mb-1">
          <span>Health</span>
          <span>{component.health_score}%</span>
        </div>
        <div className="h-2 bg-gray-200 rounded-full overflow-hidden">
          <div
            className={`h-full ${statusColors[component.status]}`}
            style={{ width: `${component.health_score}%` }}
          />
        </div>
      </div>

      {component.predicted_failure_days && (
        <div className="text-sm text-gray-600">
          Predicted service in {component.predicted_failure_days} days
        </div>
      )}

      <div className="text-xs text-gray-500 mt-2">
        Last serviced: {new Date(component.last_service).toLocaleDateString()}
      </div>
    </div>
  )
}
```

Timeline Scrubber:

```
// components/TimelineScrubber.tsx
'use client'
```

```
import { useState, useEffect } from 'react'
import { Line } from 'react-chartjs-2'

export function TimelineScrubber({ vehicleData, onDateChange }) {
  const [selectedDate, setSelectedDate] = useState(new Date())

  // Historical data from last 90 days
  const historicalData = vehicleData.historical_health_scores

  return (
    <div className="p-4 bg-gray-50 rounded-lg">
      <div className="mb-4">
        <h3 className="text-sm font-semibold mb-2">Health History</h3>
        <Line
          data={{
            labels: historicalData.map(d => d.date),
            datasets: [
              {
                label: 'Overall Health',
                data: historicalData.map(d => d.health_score),
                borderColor: 'rgb(59, 130, 246)',
                tension: 0.4
              }
            ]
          }}
          options={{
            responsive: true,
            interaction: {
              intersect: false,
              mode: 'index'
            },
            plugins: {
              tooltip: {
                callbacks: {
                  afterLabel: (context) => {
                    const date = historicalData[context.dataIndex].date
                    return `Click to view details for ${date}`
                  }
                }
              }
            },
            onClick: (event, elements) => {
              if (elements.length > 0) {
                const index = elements[0].index
```

```
            onDateChange(historicalData[index].date)
          }
        }
      }}
    />
  </div>

  <input
    type="range"
    min={0}
    max={historicalData.length - 1}
    value={historicalData.findIndex(d => d.date === selectedDate)}
    onChange={(e) => {
      const date = historicalData[parseInt(e.target.value)].date
      setSelectedDate(date)
      onDateChange(date)
    }}
    className="w-full"
  />
  </div>
 )
}
```

3.3.3 Trip Planner Page

Features:

- Route input (origin → destination)
- Estimated distance and duration
- Pre-trip safety check with component-by-component analysis
- Recommended services before departure
- Nearby service center map

```
// app/vehicles/[vin]/trip-planner/page.tsx
'use client'

import { useState } from 'react'
import { MapPin, Calendar, Users, AlertTriangle } from 'lucide-react'

export default function TripPlanner({ params }) {
  const [trip, setTrip] = useState({
    origin: '',
    destination: '',
    date: '',
```

```
  passengers: 1
})

const [analysis, setAnalysis] = useState(null)

const analyzeTripSafety = async () => {
  const response = await fetch(`/api/vehicles/${params.vin}/trip-check`, {
    method: 'POST',
    body: JSON.stringify(trip)
  })
  const data = await response.json()
  setAnalysis(data)
}

return (
  <div className="max-w-4xl mx-auto p-6">
    <h1 className="text-3xl font-bold mb-6">Plan Your Trip</h1>

    <div className="bg-white rounded-lg shadow p-6 mb-6">
      <div className="grid grid-cols-2 gap-4">
        <div>
          <label className="block text-sm font-medium mb-2">
            <MapPin className="inline w-4 h-4 mr-1" />
            From
          </label>
          <input
            type="text"
            placeholder="Bangalore"
            className="w-full border rounded px-3 py-2"
            value={trip.origin}
            onChange={(e) => setTrip({...trip, origin: e.target.value})}
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">
            <MapPin className="inline w-4 h-4 mr-1" />
            To
          </label>
          <input
            type="text"
            placeholder="Goa"
            className="w-full border rounded px-3 py-2"
            value={trip.destination}
```

```
            onChange={(e) => setTrip({...trip, destination: e.target.value})}
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">
            <Calendar className="inline w-4 h-4 mr-1" />
            Departure Date
          </label>
          <input
            type="date"
            className="w-full border rounded px-3 py-2"
            value={trip.date}
            onChange={(e) => setTrip({...trip, date: e.target.value})}
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">
            <Users className="inline w-4 h-4 mr-1" />
            Passengers
          </label>
          <input
            type="number"
            min={1}
            max={7}
            className="w-full border rounded px-3 py-2"
            value={trip.passengers}
            onChange={(e) => setTrip({...trip, passengers: parseInt(e.target.value)})}
          />
        </div>
      </div>

      <button
        onClick={analyzeTripSafety}
        className="mt-4 w-full bg-blue-600 text-white py-3 rounded-lg font-semibold
hover:bg-blue-700"
      >
        Check Trip Safety
      </button>
    </div>

    {analysis && (
      <div className="space-y-4">
```

```
          {!analysis.trip_safe && (
            <div className="bg-red-50 border-l-4 border-red-500 p-4 rounded">
              <div className="flex items-center mb-2">
                <AlertTriangle className="w-5 h-5 text-red-500 mr-2" />
                <h3 className="font-semibold text-red-800">Action Required Before Trip</h3>
              </div>
              <p className="text-sm text-red-700">
                We've identified issues that should be addressed for a safe journey.
              </p>
            </div>
          )}

          <div className="bg-white rounded-lg shadow p-6">
            <h3 className="font-semibold mb-4">Recommended Services</h3>
            {analysis.recommendations.map((rec, idx) => (
              <div key={idx} className="border-l-4 border-yellow-500 pl-4 mb-4">
                <div className="flex items-center justify-between">
                  <div>
                    <h4 className="font-medium">{rec.component}</h4>
                    <p className="text-sm text-gray-600">{rec.issue}</p>
                  </div>
                  <span className={`px-3 py-1 rounded text-sm ${
                    rec.priority === 'HIGH' ? 'bg-red-100 text-red-800' : 'bg-yellow-100 text-yellow-800'
                  }`}>
                    {rec.priority}
                  </span>
                </div>
              </div>
            ))}
          </div>

          <div className="bg-white rounded-lg shadow p-6">
            <h3 className="font-semibold mb-4">Nearby Service Centers</h3>
            {/* Map component showing service centers */}
            {/* List of centers with available slots */}
          </div>
        </div>
      )}
    </div>
  )
}
```

3.3.4 Dashboard Page

Fleet Overview (for multi-vehicle users):

```
// app/dashboard/page.tsx
export default async function Dashboard() {
  const vehicles = await getUserVehicles()

  return (
    <div className="p-6">
      <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
        <StatCard
          title="Vehicles Monitored"
          value={vehicles.length}
          icon={<Car />}
        />
        <StatCard
          title="Upcoming Services"
          value={vehicles.filter(v => v.next_service_days <= 15).length}
          icon={<Wrench />}
        />
        <StatCard
          title="Health Score"
          value={`${Math.round(vehicles.reduce((acc, v) => acc + v.health_score, 0) /
vehicles.length)}%`}
          icon={<Heart />}
        />
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {vehicles.map(vehicle => (
          <VehicleCard key={vehicle.vin} vehicle={vehicle} />
        ))}
      </div>
    </div>
  )
}
```

3.3.5 Admin Dashboard

Agent Monitoring (UEBA):

```
// app/admin/agents/page.tsx
'use client'

import { useEffect, useState } from 'react'
```

```
import { AlertTriangle, CheckCircle, XCircle } from 'lucide-react'

export default function AgentMonitoring() {
  const [agents, setAgents] = useState([])
  const [alerts, setAlerts] = useState([])

  useEffect(() => {
    // Real-time updates via WebSocket
    const ws = new WebSocket('wss://api.autocare.com/admin/agents/stream')

    ws.onmessage = (event) => {
      const data = JSON.parse(event.data)
      if (data.type === 'AGENT_STATUS') {
        setAgents(prev => updateAgentStatus(prev, data.payload))
      } else if (data.type === 'UEBA_ALERT') {
        setAlerts(prev => [data.payload, ...prev].slice(0, 50))
      }
    }

    return () => ws.close()
  }, [])

  return (
    <div className="p-6">
      <h1 className="text-3xl font-bold mb-6">Agent Monitoring</h1>

      {/* Live alerts */}
      {alerts.length > 0 && (
        <div className="bg-red-50 border-l-4 border-red-500 p-4 mb-6 rounded">
          <h3 className="font-semibold text-red-800 mb-2">
            <AlertTriangle className="inline w-5 h-5 mr-2" />
            Recent UEBA Alerts
          </h3>
          {alerts.slice(0, 3).map(alert => (
            <div key={alert.id} className="text-sm text-red-700 mb-1">
              {alert.timestamp} - {alert.agent_name}: {alert.violation}
            </div>
          ))}
        </div>
      )}

      {/* Agent status grid */}
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        {agents.map(agent => (
```

```
        <div key={agent.id} className="bg-white rounded-lg shadow p-6">
          <div className="flex items-center justify-between mb-4">
            <h3 className="font-semibold">{agent.name}</h3>
            {agent.status === 'ACTIVE' ? (
              <CheckCircle className="w-5 h-5 text-green-500" />
            ) : (
              <XCircle className="w-5 h-5 text-red-500" />
            )}
          </div>

          <div className="space-y-2 text-sm">
            <div className="flex justify-between">
              <span className="text-gray-600">Tasks Processed</span>
              <span className="font-medium">{agent.tasks_today}</span>
            </div>
            <div className="flex justify-between">
              <span className="text-gray-600">Success Rate</span>
              <span className="font-medium">{agent.success_rate}%</span>
            </div>
            <div className="flex justify-between">
              <span className="text-gray-600">Anomaly Score</span>
              <span className={`font-medium ${
                agent.anomaly_score > 75 ? 'text-red-600' : 'text-green-600'
              }`}>
                {agent.anomaly_score}/100
              </span>
            </div>
          </div>

          <div className="mt-4 pt-4 border-t">
            <div className="text-xs text-gray-500">
              Last active: {agent.last_active}
            </div>
          </div>
        </div>
      ))}
    </div>
  </div>
 )
}
```

3.4 State Management

Zustand Store Structure:

```
// store/vehicleStore.ts
import { create } from 'zustand'

interface VehicleState {
  vehicles: Vehicle[]
  selectedVehicle: Vehicle | null
  predictions: Prediction[]
  loading: boolean
  error: string | null

  // Actions
  fetchVehicles: () => Promise<void>
  selectVehicle: (vin: string) => void
  updateVehicleStatus: (vin: string, status: HealthStatus) => void
}

export const useVehicleStore = create<VehicleState>((set, get) => ({
  vehicles: [],
  selectedVehicle: null,
  predictions: [],
  loading: false,
  error: null,

  fetchVehicles: async () => {
    set({ loading: true })
    try {
      const response = await fetch('/api/vehicles')
      const vehicles = await response.json()
      set({ vehicles, loading: false })
    } catch (error) {
      set({ error: error.message, loading: false })
    }
  },

  selectVehicle: (vin) => {
    const vehicle = get().vehicles.find(v => v.vin === vin)
    set({ selectedVehicle: vehicle })
  },

  updateVehicleStatus: (vin, status) => {
    set(state => ({
      vehicles: state.vehicles.map(v =>
```

```
        v.vin === vin ? { ...v, health_status: status } : v
      )
    }))
  }
}))
```

3.5 Performance Optimization

3D Asset Optimization:

- Use Draco compression for GLTF models (70-90% size reduction)
- Implement LOD (Level of Detail) for complex models
- Use texture atlases to reduce draw calls
- Implement progressive loading (low-res → high-res)
- Use basis/KTX2 compressed textures

Code Splitting:

- Route-based code splitting (Next.js automatic)
- Dynamic imports for heavy components
- Lazy load 3D models and charts
- Defer non-critical JavaScript

Image Optimization:

- Next.js Image component with automatic WebP/AVIF
- Responsive images with srcset
- Lazy loading below-the-fold images
- CDN delivery via CloudFront

Caching Strategy:

- Static pages cached at CDN edge
- API responses cached with SWR/React Query
- Service worker for offline support
- Optimistic UI updates

---

4. AGENTS & AI SYSTEMS

4.1 Agent Architecture

Agent Design Principles:

1. **Single Responsibility:** Each agent has one primary function
2. **Autonomy:** Agents make decisions within their domain without constant human intervention
3. **Observability:** All actions logged and monitorable
4. **Graceful Degradation:** Fallback to human operators when confidence is low
5. **Continuous Learning:** Agents improve from feedback loops

4.2 Master Agent Implementation

Core Responsibilities:

- Central orchestrator coordinating all worker agents
- Maintains global state for all vehicles
- Enforces business rules and policies
- Routes tasks based on priority and agent availability
- Implements UEBA security checks
- Handles escalations

Implementation Details:

```
# master_agent/orchestrator.py
class MasterAgent:
    def __init__(self):
        self.redis = Redis(host='redis-cluster', decode_responses=True)
        self.rabbitmq = RabbitMQConnection()
        self.ueba = UEBASecurityLayer()

        # Worker agent registry
        self.workers = {
            'data_analysis': DataAnalysisAgent(),
            'diagnosis': DiagnosisAgent(),
            'customer_engagement': CustomerEngagementAgent(),
            'scheduling': SchedulingAgent(),
            'feedback': FeedbackAgent(),
            'quality_insights': QualityInsightsModule()
        }

    async def process_vehicle_event(self, vin: str, telemetry: dict):
        """Main event processing pipeline"""

        # Step 1: Log event
        event_id = await self.log_event(vin, telemetry)

        # Step 2: Security check
        if not await self.ueba.validate_data_source(telemetry):
```

```
            await self.raise_security_alert("Anomalous telemetry source", vin)
            return

        # Step 3: Update vehicle state
        await self.update_vehicle_state(vin, telemetry)

        # Step 4: Check if analysis needed
        if await self.should_analyze(vin):
            task = self.create_analysis_task(vin, telemetry)
            await self.rabbitmq.publish('data_analysis_queue', task)

    async def handle_prediction(self, prediction: dict):
        """Handle prediction from Diagnosis Agent"""

        vin = prediction['vin']
        confidence = prediction['confidence']
        priority = prediction['priority']

        # Business rules
        if confidence >= 0.85 and priority in ['P1', 'P2']:
            # High confidence, urgent: Initiate customer engagement
            await self.initiate_customer_engagement(prediction)

        elif confidence >= 0.70:
            # Medium confidence: Enhanced monitoring
            await self.schedule_enhanced_monitoring(vin)

        elif confidence >= 0.50:
            # Low confidence: Log for pattern analysis
            await self.log_low_confidence(prediction)

        else:
            # Very low confidence: Discard
            pass

    async def initiate_customer_engagement(self, prediction: dict):
        """Start customer engagement workflow"""

        vin = prediction['vin']

        # Get customer profile
        customer = await self.get_customer_profile(vin)

        # Check communication preferences
```

```python
        if not customer.consent_voice_call:
            # Fall back to app notification
            await self.send_app_notification(customer, prediction)
            return

        # Check do-not-disturb hours
        if await self.is_dnd_time(customer):
            # Schedule for later
            await self.schedule_delayed_engagement(customer, prediction)
            return

        # Create engagement task
        task = {
            'task_id': generate_uuid(),
            'vin': vin,
            'customer': customer.to_dict(),
            'prediction': prediction,
            'priority': prediction['priority'],
            'attempt': 1,
            'max_attempts': 3
        }

        # Publish to engagement queue
        await self.rabbitmq.publish(
            'customer_engagement_queue',
            task,
            priority=self.get_priority_level(prediction['priority'])
        )

        # Update state
        await self.update_vehicle_state(vin, {
            'engagement_status': 'INITIATED',
            'engagement_task_id': task['task_id']
        })
```

State Machine Implementation:

```python
# master_agent/state_machine.py
from enum import Enum
from typing import Dict, Callable

class VehicleState(Enum):
    HEALTHY = "HEALTHY"
    MONITORING = "MONITORING"
```

```python
    PREDICTION_CONFIRMED = "PREDICTION_CONFIRMED"
    CUSTOMER_CONTACTED = "CUSTOMER_CONTACTED"
    APPOINTMENT_SCHEDULED = "APPOINTMENT_SCHEDULED"
    SERVICE_IN_PROGRESS = "SERVICE_IN_PROGRESS"
    SERVICE_COMPLETED = "SERVICE_COMPLETED"
    FEEDBACK_COLLECTED = "FEEDBACK_COLLECTED"

class StateMachine:
    def __init__(self):
        self.transitions = {
            VehicleState.HEALTHY: {
                'anomaly_detected': VehicleState.MONITORING
            },
            VehicleState.MONITORING: {
                'prediction_confirmed': VehicleState.PREDICTION_CONFIRMED,
                'false_alarm': VehicleState.HEALTHY
            },
            VehicleState.PREDICTION_CONFIRMED: {
                'customer_contacted': VehicleState.CUSTOMER_CONTACTED,
                'contact_failed': VehicleState.PREDICTION_CONFIRMED  # Retry
            },
            VehicleState.CUSTOMER_CONTACTED: {
                'appointment_booked': VehicleState.APPOINTMENT_SCHEDULED,
                'customer_declined': VehicleState.HEALTHY,  # Log but move on
                'callback_requested': VehicleState.CUSTOMER_CONTACTED  # Retry later
            },
            VehicleState.APPOINTMENT_SCHEDULED: {
                'service_started': VehicleState.SERVICE_IN_PROGRESS,
                'appointment_cancelled': VehicleState.PREDICTION_CONFIRMED  # Reschedule
            },
            VehicleState.SERVICE_IN_PROGRESS: {
                'service_completed': VehicleState.SERVICE_COMPLETED
            },
            VehicleState.SERVICE_COMPLETED: {
                'feedback_received': VehicleState.FEEDBACK_COLLECTED
            },
            VehicleState.FEEDBACK_COLLECTED: {
                'cycle_reset': VehicleState.HEALTHY
            }
        }

    async def transition(self, vin: str, event: str) -> bool:
        """Attempt state transition"""
        current_state = await self.get_current_state(vin)
```

```
        if event not in self.transitions[current_state]:
            logger.warning(f"Invalid transition: {current_state} -> {event}")
            return False

        new_state = self.transitions[current_state][event]

        # Update state in Redis
        await self.update_state(vin, new_state)

        # Log transition
        await self.log_transition(vin, current_state, new_state, event)

        # Trigger side effects
        await self.handle_state_change(vin, new_state)

        return True
```

4.3 Data Analysis Agent

Purpose: Real-time anomaly detection from streaming telemetry

Implementation:

```
# agents/data_analysis_agent.py
import numpy as np
from sklearn.ensemble import IsolationForest
from kafka import KafkaConsumer
import asyncio

class DataAnalysisAgent:
    def __init__(self):
        self.consumer = KafkaConsumer(
            'vehicle.telemetry.raw',
            bootstrap_servers='kafka-cluster:9092',
            group_id='data_analysis_group',
            value_deser
```