IoT Polyglot Portfolio Project - Complete Requirements Document

Project Overview

Project Name: HomeGuard IoT Platform **Purpose:** A production-grade portfolio project demonstrating polyglot persistence, event-driven architecture, cloud-native patterns, and **Agentic AI** for intelligent IoT device management.

Deployment Target: Rancher (local Kubernetes cluster) **Demo Strategy:** Pre-seeded users with device simulator + scenario engine generating real-time events, with AI-powered conversational interface

---

Technology Stack

Languages

Language	Use Case
Go	All backend services, API Gateway, Device Simulator, Scenario Engine
Python	Analytics Service, Anomaly ML Service, Agentic AI Service, MCP Server
TypeScript/React	Frontend UI with Chat Interface

Data Stores

Technology	Purpose	Data Type
PostgreSQL	User accounts, authentication, subscriptions	Relational, transactional
TimescaleDB	Sensor readings, time-series analytics	Time-series (PostgreSQL extension)
ScyllaDB	High-volume IoT events (motion, door, heartbeats)	Wide-column, write-heavy
MongoDB	Device configurations, automation rules, agent memory	Document store, flexible schema
Redis	Current device state, caching, pub/sub	Key-value, real-time
Kafka	Event streaming, ingestion bus	Event log, decoupling

AI/ML Stack

Technology	Purpose
Google Gemini API	LLM for agentic reasoning, natural language understanding, conversation
scikit-learn	Anomaly detection (IsolationForest), predictive maintenance
MCP Server	Model Context Protocol - exposes tools for LLM to query/control system

Observability Stack

Technology	Purpose
Grafana	Dashboards, visualization, alerting, escalations
Prometheus	Metrics collection from all services
Loki	Log aggregation, event search

Workflow Automation

Technology	Purpose
n8n	User-defined automations, scheduled jobs, complex workflows

---

Architecture Overview

```
┌──────────────────────────────────────────────────────────────────
│                    RANCHER (Local K8s)                    │
├──────────────────────────────────────────────────────────────────
│                                                 │
│
┌──────────────────────────────────────────────────────────────────
│
│  │                 React UI (TypeScript)                │ │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
┌──────────────────────────────────────────┐ │ │
│  │  │ Dashboard │ │ Devices │ │ Analytics │ │ 💬 AI Chat Interface │ │ │
│  │  └──────────────┘  └──────────────┘  └──────────────┘
└──────────────────────────────────────────┘ │ │
│  │
┌──────────────────────────────────────────────────────────────────
│ │
│  │  │ 🎬 Demo Scenario Panel                          │ │ │
│  │  │ [🏖 Vacation] [🎉 Party] [🚨 Break-in] [🔋 Low Battery] [📦 Package]     │ │ │
│  │
└──────────────────────────────────────────────────────────────────
│ │
│
└──────────────────────────────────────────────────────┬───────────
│
│                       │                      │
│                       ▼                       │
│
```

```
┌──────────────────────────────────────────────────────────────────┐
│                                                                  │
│  │                 API Gateway (Go)                       │  │
│  │   • Rate Limiting  • Circuit Breaker  • JWT Auth  • Routing   │  │
│                                                                  │
└──────────────────────────────┬───────────────────────────────────┘
│
│              │                    │
│
┌────────┬────────┬────────┬───┴────┬────────┬────────┬────────┬────────┐
│
│  ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼  │
│ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐ ┌──────┐
┌────────┐ │
│ │User  │ │Device│ │Device│ │Event │ │Notif │ │Analyt│ │Anomal│ │Agent │ │Scenario│ │
│ │Svc   │ │Svc   │ │Ingest│ │Proc  │ │Svc   │ │ics   │ │y ML  │ │AI    │ │Engine  │ │
│ │(Go)  │ │(Go)  │ │(Go)  │ │(Go)  │ │(Go)  │ │(Py)  │ │(Py)  │ │(Py)  │ │(Go)    │ │
│ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘ └──┬───┘
└───┬────┘ │
│  │   │   │   │   │   │   │   │    │   │
│  │   │   │   │   │   │   │   ▼    │   │
│  │   │   │   │   │   │   │  ┌────────┐  │   │
│  │   │   │   │   │   │   │  │  MCP   │  │   │
│  │   │   │   │   │   │   └─►│ Server │  │   │
│  │   │   │   │   │   │     │ (Py)   │  │   │
│  │   │   │   │   │   │     └───┬────┘  │   │
│  │   │   │   │   │   │         │    │   │
│  │   │   │   │   │   │         ▼    │   │
│  │   │   │   │   │   │   ┌─────────────┐  │   │
│  │   │   │   │   │   │   │ Google      │  │   │
│  │   │   │   │   │   │   │ Gemini API  │  │   │
│  │   │   │   │   │   │   │ (External)  │  │   │
│  │   │   │   │   │   │   └─────────────┘  │   │
│  │   │   │   │   │   │              │   │
│  │   │   │   │   ▼   │              │   │
│  │   │   │   │ ┌────────┐ │              │   │
│  │   │   │   └►│  n8n   │◄┘              │   │
│  │   │   │    │Workflows│              │   │
│  │   │   │    └─────────┘              │   │
│  │   │   │                         │   │
│  ▼   ▼   ▼      ▼      ▼     ▼    ▼    │
│ ┌──────┐ ┌──────┐ ┌────────┐ ┌────────┐ ┌──────────┐ ┌──────────┐   │
│ │Postgr│ │Mongo │ │ Kafka  │ │ScyllaDB│ │Redis │ │Timescale│   │
│ │ es   │ │ DB   │ │        │ │        │ │ DB   │ │         │
```

```
│ └─────┘ └─────┘ └───────┘ └───────┘ └─────┘ └───────┘      │
│                        │
│
┌──────────────────────────────────────────────────────────
│
│ │          Grafana + Prometheus + Loki           │ │
│ │   (Dashboards, Metrics, Logs, Alerting, Escalations)   │ │
│
└──────────────────────────────────────────────────────────
│
│                        │
│
┌──────────────────────────────────────────────────────────
│
│ │          Device Simulator (Go)           │ │
│ │     Generates realistic IoT events → Kafka      │ │
│
└──────────────────────────────────────────────────────────
│
│                        │
└──────────────────────────────────────────────────────────
```

Microservices Specification

1. API Gateway (Go)

Responsibilities:

- Request routing to backend services
- JWT authentication/authorization
- Rate limiting (per user, per endpoint)
- Circuit breaker for downstream services
- Request/response logging
- CORS handling

Endpoints:

```
POST   /auth/login             → User Service
GET    /users/me               → User Service
GET    /devices                → Device Service
GET    /devices/{id}           → Device Service
POST   /devices/{id}/command   → Device Ingest Service
GET    /events                 → Event Processor
GET    /events/stream          → WebSocket (Notification Service)
GET    /analytics/*            → Analytics Service
POST   /automations            → Device Service (stores) + n8n (executes)
GET    /automations            → Device Service

# AI Endpoints
POST   /ai/chat                → Agentic AI Service
GET    /ai/chat/history        → Agentic AI Service
POST   /ai/chat/stream         → WebSocket for streaming responses

# Demo Endpoints
POST   /demo/scenarios/{name}/start  → Scenario Engine
POST   /demo/scenarios/{name}/stop   → Scenario Engine
POST   /demo/events                  → Scenario Engine (single event)
GET    /demo/scenarios               → Scenario Engine (list all)
```

Tech:

- Framework: Chi or Gin
- Rate Limiter: golang.org/x/time/rate + Redis for distributed
- Circuit Breaker: sony/gobreaker
- Metrics: prometheus/client_golang

---

2. User Service (Go)

Responsibilities:

- User authentication (login only, no registration for demo)
- JWT token generation/validation
- User profile management
- Pre-seeded demo users

Database: PostgreSQL

Schema:

```sql
CREATE TABLE users (
    id            UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email         VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    name          VARCHAR(255) NOT NULL,
    role          VARCHAR(50) DEFAULT 'user',  -- user, admin
    created_at    TIMESTAMPTZ DEFAULT NOW(),
    updated_at    TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE user_sessions (
    id            UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id       UUID REFERENCES users(id),
    token_hash    VARCHAR(255) NOT NULL,
    expires_at    TIMESTAMPTZ NOT NULL,
    created_at    TIMESTAMPTZ DEFAULT NOW()
);
```

Pre-seeded Users:

```sql
INSERT INTO users (email, password_hash, name, role) VALUES
('john@demo.com', '<bcrypt_hash>', 'John Smith', 'user'),
('sarah@demo.com', '<bcrypt_hash>', 'Sarah Johnson', 'user'),
('admin@demo.com', '<bcrypt_hash>', 'Admin User', 'admin');
```

---

3. Device Service (Go)

Responsibilities:

- Device CRUD operations
- Device configuration management
- Automation rules management
- Current device state (via Redis)

Databases:

- MongoDB: Device configurations, automation rules
- Redis: Current device state cache

MongoDB Collections:

```javascript
```

```
// devices collection
{
  "_id": ObjectId,
  "device_id": "device-uuid",
  "user_id": "user-uuid",
  "name": "Front Door Lock",
  "type": "lock",  // lock, motion, thermostat, door, camera, glass_break, smoke
  "model": "Yale Assure Lock 2",
  "firmware_version": "1.2.3",
  "location": {
    "zone": "Front Entrance",
    "floor": 1
  },
  "config": {
    "auto_lock_delay": 30,
    "volume": "medium"
  },
  "created_at": ISODate,
  "updated_at": ISODate
}

// automations collection
{
  "_id": ObjectId,
  "user_id": "user-uuid",
  "name": "Night Motion Alert",
  "enabled": true,
  "trigger": {
    "type": "device_event",
    "device_id": "device-uuid",
    "event_type": "motion_detected",
    "conditions": {
      "time_range": {"start": "22:00", "end": "06:00"},
      "arm_state": "armed"
    }
  },
  "actions": [
    {"type": "notification", "channel": "push", "message": "Motion detected at night!"},
    {"type": "webhook", "url": "n8n-webhook-url"}
  ],
  "created_at": ISODate,
```

```
  "updated_at": ISODate
}
```

Redis Keys (Device State):

```
device:state:{device_id}      → Hash: current state
device:online:{device_id}     → String: last heartbeat timestamp
user:devices:{user_id}        → Set: device IDs for user
```

---

4. Device Ingest Service (Go)

Responsibilities:

- Receive device commands from API
- Produce events to Kafka
- Handle device heartbeats
- Validate event payloads

Kafka Topics Produced:

- `device.commands` - Commands sent to devices
- `device.events` - Sensor events (motion, door, etc.)
- `device.heartbeats` - Device health checks
- `device.alerts` - Anomaly events

Event Schema (JSON):

```
json
```

```json
{
  "event_id": "uuid",
  "device_id": "uuid",
  "user_id": "uuid",
  "event_type": "motion_detected",
  "payload": {
    "zone": "living_room",
    "intensity": 0.85,
    "signature": "human"
  },
  "timestamp": "2024-01-15T14:32:05.123Z",
  "metadata": {
    "firmware_version": "1.2.3",
    "signal_strength": -45,
    "is_simulated": false
  }
}
```

---

5. Event Processor (Go)

Responsibilities:

- Consume events from Kafka
- Write raw events to ScyllaDB
- Update device state in Redis
- Trigger n8n webhooks for automation rules
- Publish to Redis pub/sub for real-time UI
- Forward events to Anomaly ML Service for scoring

Kafka Topics Consumed:

- `device.events`
- `device.heartbeats`
- `device.alerts`

ScyllaDB Schema:

```cql
```

```
CREATE KEYSPACE iot_events WITH replication = {
    'class': 'SimpleStrategy',
    'replication_factor': 1
};

-- Events by device (for device history)
CREATE TABLE iot_events.events_by_device (
    device_id      UUID,
    event_date     DATE,
    event_time     TIMESTAMP,
    event_id       UUID,
    event_type     TEXT,
    payload        TEXT,  -- JSON
    PRIMARY KEY ((device_id, event_date), event_time, event_id)
) WITH CLUSTERING ORDER BY (event_time DESC, event_id ASC);

-- Events by user (for user's event feed)
CREATE TABLE iot_events.events_by_user (
    user_id        UUID,
    event_date     DATE,
    event_time     TIMESTAMP,
    event_id       UUID,
    device_id      UUID,
    event_type     TEXT,
    payload        TEXT,
    PRIMARY KEY ((user_id, event_date), event_time, event_id)
) WITH CLUSTERING ORDER BY (event_time DESC, event_id ASC);

-- Events by type (for analytics)
CREATE TABLE iot_events.events_by_type (
    event_type     TEXT,
    event_date     DATE,
    event_time     TIMESTAMP,
    event_id       UUID,
    device_id      UUID,
    user_id        UUID,
    payload        TEXT,
    PRIMARY KEY ((event_type, event_date), event_time, event_id)
) WITH CLUSTERING ORDER BY (event_time DESC, event_id ASC);
```

6. Notification Service (Go)

Responsibilities:

- Subscribe to Redis pub/sub channels
- Maintain WebSocket connections to UI clients
- Push real-time updates to connected clients
- Route notifications by user

Redis Pub/Sub Channels:

```
events:user:{user_id}       → Events for specific user
events:global               → System-wide events (admin)
device:state:{device_id}    → Device state changes
agent:response:{user_id}    → AI agent responses
```

WebSocket Protocol:

```json
// Client subscribes
{"action": "subscribe", "channels": ["events", "devices", "agent"]}

// Server pushes event
{
  "type": "device_event",
  "device_id": "uuid",
  "event_type": "motion_detected",
  "timestamp": "2024-01-15T14:32:05.123Z",
  "data": {...}
}

// Server pushes agent response
{
  "type": "agent_response",
  "message": "I've locked the front door for you.",
  "actions_taken": ["lock_front_door"],
  "timestamp": "2024-01-15T14:32:05.123Z"
}
```

---

7. Analytics Service (Python)

Responsibilities:

- Time-series aggregations and queries
- Generate reports and trends
- Serve analytics API endpoints
- Feed data to Grafana dashboards

Database: TimescaleDB (PostgreSQL + extension)

TimescaleDB Schema:

```sql
```

```sql
-- Enable TimescaleDB
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Sensor readings (hypertable)
CREATE TABLE sensor_readings (
    time           TIMESTAMPTZ NOT NULL,
    device_id      UUID NOT NULL,
    user_id        UUID NOT NULL,
    reading_type   VARCHAR(50) NOT NULL,
    value          DOUBLE PRECISION NOT NULL,
    unit           VARCHAR(20)
);

SELECT create_hypertable('sensor_readings', 'time');

-- Create indexes
CREATE INDEX idx_sensor_device ON sensor_readings (device_id, time DESC);
CREATE INDEX idx_sensor_user ON sensor_readings (user_id, time DESC);

-- Event aggregations (continuous aggregate)
CREATE MATERIALIZED VIEW hourly_events
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 hour', time) AS bucket,
    device_id,
    reading_type,
    COUNT(*) as event_count,
    AVG(value) as avg_value
FROM sensor_readings
GROUP BY bucket, device_id, reading_type;

-- Enable compression
ALTER TABLE sensor_readings SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'device_id'
);

SELECT add_compression_policy('sensor_readings', INTERVAL '7 days');
SELECT add_retention_policy('sensor_readings', INTERVAL '90 days');
```

---

8. Device Simulator (Go)

Responsibilities:

- Generate realistic IoT events continuously
- Simulate device behaviors (motion patterns, temperature cycles)
- Produce events to Kafka
- Configurable event rate

Simulation Patterns:

```go
// Motion sensors: Random triggers, more frequent during "day" hours
// Thermostats: Gradual temperature changes, cycles
// Door locks: Lock/unlock patterns (morning/evening peaks)
// Heartbeats: Every 30 seconds per device
// Anomalies: 1% chance of battery_low, offline, etc.
```

Configuration:

```yaml
simulator:
  event_rate: 10  # events per second across all devices
  devices_per_user:
    john@demo.com: 5
    sarah@demo.com: 12
  anomaly_rate: 0.01
  patterns:
    motion:
      peak_hours: [8, 9, 17, 18, 19, 20]
      frequency: high
    thermostat:
      min_temp: 68
      max_temp: 76
      cycle_minutes: 30
```

---

9. Scenario Engine (Go) - NEW

Responsibilities:

- Parse and execute demo scenario definitions

- Schedule events with precise timing
- Inject events into Kafka (same pipeline as real events)
- Support time manipulation for demos
- Provide API for UI to trigger scenarios

API Endpoints:

```
POST /scenarios/{name}/start    - Start a scenario
POST /scenarios/{name}/stop     - Stop a running scenario
POST /events                    - Inject single event
POST /time                      - Set simulated time (for demos)
GET  /scenarios                 - List all available scenarios
GET  /scenarios/{name}/status   - Check if running
```

Scenario Definition (YAML):

```yaml
```

```yaml
# scenarios/break_in.yaml
name: break_in
description: "Simulates a break-in through back window"
duration: 30s
events:
  - at: 0s
    type: glass_break
    device: back_window_sensor
    payload:
      intensity: 0.92

  - at: 2s
    type: motion_detected
    device: bedroom_motion
    payload:
      zone: bedroom
      confidence: 0.88
      signature: human

  - at: 5s
    type: motion_detected
    device: hallway_motion
    payload:
      zone: hallway

  - at: 8s
    type: door_open
    device: back_door

  - at: 10s
    type: motion_stopped

expected_agent_behavior:
  - should_alert: true
  - should_notify_monitoring: true
  - should_document_incident: true
```

Pre-Built Scenarios:

Scenario	Description	Events
`vacation_mode`	User announces vacation	Chat trigger → Agent creates plan
`break_in`	Full break-in sequence	Glass break → motion → door
`false_alarm_pet`	2 AM motion (pet)	Agent should suppress alert
`guest_arrival`	Expected guest comes	Doorbell → face recognition → auto-unlock
`low_battery`	Battery drops to 15%	Agent predicts replacement
`unusual_pattern`	Door unlock, no follow-up motion	Agent investigates
`party_mode`	User announces party	Agent adjusts sensitivity
`fire_emergency`	Smoke + heat anomaly	Full emergency response
`package_delivery`	Doorbell + person + package	Agent identifies, tracks
`wellness_check`	No activity from profile	Agent escalates
`energy_insight`	30 days of patterns analyzed	Agent suggests optimizations
`day_in_life`	24hr compressed to 5min	Shows agent learning patterns

---

10. Anomaly ML Service (Python) - NEW

Responsibilities:

- Train anomaly detection models on normal patterns
- Score incoming events for anomalies
- Predictive maintenance (battery life, device health)
- Pattern learning and baseline establishment

ML Models:

```python
```

```python
from sklearn.ensemble import IsolationForest
from sklearn.linear_model import LinearRegression

# Anomaly Detection
class AnomalyDetector:
    def __init__(self):
        self.model = IsolationForest(contamination=0.01, random_state=42)

    def train(self, normal_events):
        features = self.extract_features(normal_events)
        self.model.fit(features)

    def score(self, event):
        features = self.extract_features([event])
        prediction = self.model.predict(features)
        return prediction[0] == -1  # -1 = anomaly

# Predictive Maintenance
class BatteryPredictor:
    def __init__(self):
        self.model = LinearRegression()

    def predict_days_remaining(self, device_id, readings):
        # Predict when battery will hit 10%
        self.model.fit(readings['days'], readings['battery_level'])
        days_until_low = (10 - self.model.intercept_) / self.model.coef_[0]
        return max(0, days_until_low)
```

Anomaly Types Detected:

- Unusual motion times (2 AM motion when no one home)
- Device behavior changes (sensor reporting unusual values)
- Pattern deviations (activity in unexpected zones)
- System anomalies (increased latency, failed events)

API Endpoints:

```
POST /anomaly/score        - Score an event
POST /anomaly/train        - Retrain models
GET  /anomaly/baseline     - Get learned baselines
GET  /predictions/battery/{device_id}  - Battery prediction
GET  /predictions/maintenance/{device_id}  - Maintenance prediction
```

---

11. Agentic AI Service (Python) - NEW

Responsibilities:

- Natural language understanding and conversation
- Contextual reasoning about home state
- Autonomous decision making
- Multi-step task execution
- Learning user preferences and patterns

LLM: Google Gemini API (key provided by user)

Architecture:

```python
```

```python
import google.generativeai as genai
from mcp import MCPClient

class AgenticAIService:
    def __init__(self, gemini_api_key: str):
        genai.configure(api_key=gemini_api_key)
        self.model = genai.GenerativeModel('gemini-1.5-flash')
        self.mcp_client = MCPClient()

    async def process_message(self, user_id: str, message: str) -> AgentResponse:
        # 1. Get context from MCP
        home_state = await self.mcp_client.call_tool("get_home_summary", {"user_id": user_id})
        recent_events = await self.mcp_client.call_tool("get_recent_events", {"user_id": user_id, "hours": 24})

        # 2. Build prompt with context
        prompt = self.build_prompt(message, home_state, recent_events)

        # 3. Get LLM response with function calling
        response = await self.model.generate_content(
            prompt,
            tools=self.get_available_tools()
        )

        # 4. Execute any tool calls
        actions_taken = await self.execute_tool_calls(response.tool_calls)

        # 5. Return response
        return AgentResponse(
            message=response.text,
            actions_taken=actions_taken,
            reasoning=response.reasoning
        )
```

System Prompt:

```
You are HomeGuard AI, an intelligent home security assistant. You have access to:
- Real-time device states (locks, sensors, thermostats, cameras)
- Event history (motion, door, alerts)
- User automation rules
- Anomaly detection insights

Your capabilities:
1. Answer questions about home status
2. Control devices (lock/unlock, arm/disarm, adjust thermostat)
3. Create and modify automation rules
4. Analyze patterns and suggest improvements
5. Respond to emergencies with coordinated actions
6. Predict maintenance needs

Always:
- Consider context before alerting (reduce false alarms)
- Explain your reasoning when taking actions
- Ask for confirmation on destructive/security-sensitive actions
- Learn from user feedback and preferences

Current home state will be provided via tools. Use them to get accurate information.
```

Conversation Memory (MongoDB):

```javascript
```

```
// agent_conversations collection
{
  "_id": ObjectId,
  "user_id": "user-uuid",
  "session_id": "session-uuid",
  "messages": [
    {
      "role": "user",
      "content": "I'm leaving for vacation tomorrow",
      "timestamp": ISODate
    },
    {
      "role": "assistant",
      "content": "I'll prepare your home for vacation mode...",
      "actions_taken": ["set_vacation_mode"],
      "timestamp": ISODate
    }
  ],
  "context": {
    "vacation_mode": true,
    "return_date": "2024-01-29"
  },
  "created_at": ISODate,
  "updated_at": ISODate
}

// agent_learnings collection (user preferences)
{
  "_id": ObjectId,
  "user_id": "user-uuid",
  "learnings": {
    "alert_preferences": {
      "suppress_pet_motion": true,
      "night_motion_threshold": 0.8
    },
    "routines": {
      "morning_departure": "07:15",
      "evening_return": "18:30"
    },
    "automation_style": "proactive"
  },
```

```
    "updated_at": ISODate
}
```

---

12. MCP Server (Python) - NEW

Responsibilities:

- Expose tools for Gemini LLM to interact with the system
- Query device states, events, analytics
- Execute actions (lock, unlock, arm, etc.)
- Provide structured data to LLM

MCP Tools Definition:

```python
```

```
tools = [
    {
        "name": "get_home_summary",
        "description": "Get current state of all devices and overall home status",
        "parameters": {
            "user_id": {"type": "string", "required": True}
        }
    },
    {
        "name": "get_device_state",
        "description": "Get current state of a specific device",
        "parameters": {
            "device_id": {"type": "string", "required": True}
        }
    },
    {
        "name": "get_recent_events",
        "description": "Get recent events for a user or device",
        "parameters": {
            "user_id": {"type": "string"},
            "device_id": {"type": "string"},
            "hours": {"type": "integer", "default": 24},
            "event_type": {"type": "string"}
        }
    },
    {
        "name": "get_anomalies",
        "description": "Get detected anomalies and unusual patterns",
        "parameters": {
            "user_id": {"type": "string", "required": True},
            "hours": {"type": "integer", "default": 24}
        }
    },
    {
        "name": "control_device",
        "description": "Send command to a device (lock, unlock, arm, etc.)",
        "parameters": {
            "device_id": {"type": "string", "required": True},
            "command": {"type": "string", "required": True},
            "parameters": {"type": "object"}
        }
    },
    {
```

```
        "name": "create_automation",
        "description": "Create a new automation rule",
        "parameters": {
            "user_id": {"type": "string", "required": True},
            "name": {"type": "string", "required": True},
            "trigger": {"type": "object", "required": True},
            "actions": {"type": "array", "required": True}
        }
    },
    {
        "name": "set_home_mode",
        "description": "Set home mode (home, away, vacation, party, sleep)",
        "parameters": {
            "user_id": {"type": "string", "required": True},
            "mode": {"type": "string", "required": True},
            "duration_hours": {"type": "integer"},
            "settings": {"type": "object"}
        }
    },
    {
        "name": "get_predictions",
        "description": "Get maintenance predictions for devices",
        "parameters": {
            "user_id": {"type": "string", "required": True},
            "device_id": {"type": "string"}
        }
    },
    {
        "name": "get_energy_insights",
        "description": "Get energy usage patterns and recommendations",
        "parameters": {
            "user_id": {"type": "string", "required": True},
            "period_days": {"type": "integer", "default": 30}
        }
    }
]
```

Tool Implementation Example:

```python
```

```python
class MCPServer:
    def __init__(self, redis_client, mongo_client, scylla_session):
        self.redis = redis_client
        self.mongo = mongo_client
        self.scylla = scylla_session

    async def get_home_summary(self, user_id: str) -> dict:
        # Get all devices for user
        device_ids = await self.redis.smembers(f"user:devices:{user_id}")

        devices = []
        for device_id in device_ids:
            state = await self.redis.hgetall(f"device:state:{device_id}")
            config = await self.mongo.devices.find_one({"device_id": device_id})
            online = await self.redis.get(f"device:online:{device_id}")

            devices.append({
                "device_id": device_id,
                "name": config["name"],
                "type": config["type"],
                "state": state,
                "online": online is not None,
                "battery": state.get("battery"),
                "last_event": state.get("last_event_time")
            })

        # Calculate overall status
        all_secure = all(
            d["state"].get("locked", True)
            for d in devices if d["type"] == "lock"
        )

        return {
            "user_id": user_id,
            "total_devices": len(devices),
            "online_devices": sum(1 for d in devices if d["online"]),
            "all_secure": all_secure,
            "devices": devices,
            "active_mode": await self.get_current_mode(user_id)
        }
```

Agentic AI Use Cases

Customer-Facing AI Use Cases

Use Case	Trigger	Agent Behavior
Vacation Mode	"I'm leaving for vacation for 2 weeks"	Creates comprehensive plan: random lights, lower HVAC, enhanced sensitivity, daily summaries
Guest Access	"My mom is coming Thursday at 2pm"	Creates temporary code, disables alerts for that window, notifies on arrival/departure
Contextual False Alarm Prevention	2 AM motion + pet registered + no door breach	Suppresses alert, logs as pet activity, learns pattern
Anomaly Investigation	Door unlocked 3x with no follow-up motion	Investigates, presents options, pulls camera footage
Energy + Security Correlation	Pattern analysis after 30 days	Suggests HVAC adjustments based on presence patterns
Emergency Coordination	Smoke + heat signature	Unlocks doors, kills HVAC, notifies contacts, shares floor plan with fire dept
Routine Learning	After observing daily patterns	Suggests automations: pre-warm bathroom, start coffee, auto-arm
Proactive Safety Checks	Before user leaves (geofence)	"Garage open for 45 min, back window open, battery low on lock"
Natural Conversation Control	"It's getting dark"	"Want me to turn on porch light and close garage?"
Maintenance Predictions	Battery/device health analysis	"Front door lock will need batteries in ~5 days"
Party Mode	"Having a party Saturday, 20 people"	Adjusts sensitivity, monitors perimeter only, resumes at 2am
Incident Documentation	Break-in detected	Full timestamped report with video, sensor data, motion path

Internal/Downstream AI Use Cases

Use Case	Service	Purpose
Anomaly Scoring	Anomaly ML Service	Score every event for unusual patterns
Predictive Maintenance	Anomaly ML Service	Predict device failures, battery depletion
Pattern Baseline	Anomaly ML Service	Learn normal behavior to detect deviations
Alert Routing	n8n + Agent	Intelligent escalation based on severity and context
False Alarm Reduction	Event Processor + Agent	Context-aware filtering before alerting

Resilience Patterns

1. Circuit Breaker

Implementation: sony/gobreaker in all Go services

Configuration:

```go
var DefaultCBConfig = CircuitBreakerConfig{
    MaxRequests: 3,
    Interval:    10 * time.Second,
    Timeout:     30 * time.Second,
    ReadyToTrip: func(counts Counts) bool {
        failureRatio := float64(counts.TotalFailures) / float64(counts.Requests)
        return counts.Requests >= 10 && failureRatio >= 0.6
    },
}
```

Circuit Breaker Locations:

Service	Protects Calls To
API Gateway	All downstream services
Device Service	MongoDB, Redis
Event Processor	ScyllaDB, Redis, n8n webhooks
Analytics Service	TimescaleDB
Agentic AI Service	Gemini API, MCP Server
Notification Service	Redis

2. Rate Limiting

Implementation: Token bucket algorithm with Redis backend

Strategies:

```go
```

```
type RateLimitConfig struct {
    UserRequestsPerMinute: 100,
    UserBurstSize:         20,

    EndpointLimits: map[string]int{
        "POST /devices/*/command": 10,
        "POST /ai/chat":           20,  // AI chat limit
        "GET /events":            60,
        "GET /events/stream":      5,
    },

    GlobalRequestsPerSecond: 1000,
}
```

3. Exception Handling

Standard Error Response:

```json
{
    "error": {
        "code": "DEVICE_NOT_FOUND",
        "message": "Device with ID xyz not found",
        "details": {"device_id": "xyz"},
        "trace_id": "abc123",
        "timestamp": "2024-01-15T14:32:05.123Z"
    }
}
```

4. Retry Policies

```go
var DefaultRetryConfig = RetryConfig{
    MaxRetries:     3,
    InitialBackoff: 100 * time.Millisecond,
    MaxBackoff:     5 * time.Second,
    BackoffFactor:  2.0,
}
```

Logging Strategy

Structured Logging Format

```json
{
  "timestamp": "2024-01-15T14:32:05.123Z",
  "level": "info",
  "service": "agentic-ai-service",
  "trace_id": "abc123",
  "user_id": "user-uuid",
  "message": "Agent processed user request",
  "data": {
    "intent": "vacation_mode",
    "actions_taken": ["set_lights_random", "lower_hvac", "enable_alerts"],
    "duration_ms": 1250,
    "llm_tokens": 450
  }
}
```

Loki Query Patterns

```logql
# AI Agent interactions
{service="agentic-ai-service"} | json | intent="vacation_mode"

# By User
{service=~".+"} |= "user_id" | json | user_id="user-uuid"

# Anomalies detected
{service="anomaly-ml-service"} |= "anomaly" | json | is_anomaly=true

# Slow AI responses
{service="agentic-ai-service"} | json | duration_ms > 2000

# Scenario executions
{service="scenario-engine"} | json | scenario_name="break_in"
```

Metrics Strategy

Prometheus Metrics

Service Metrics (all services):

```
http_requests_total{service, method, endpoint, status}
http_request_duration_seconds{service, method, endpoint}
```

AI-Specific Metrics:

```
# Gemini API metrics
gemini_requests_total{service, model, status}
gemini_request_duration_seconds{service, model}
gemini_tokens_used_total{service, type}  # input/output

# Agent metrics
agent_conversations_total{user_id, intent}
agent_actions_taken_total{action_type}
agent_false_alarm_prevented_total{}
agent_response_time_seconds{}

# Anomaly ML metrics
anomaly_events_scored_total{is_anomaly}
anomaly_model_accuracy{}
prediction_requests_total{prediction_type}
```

Scenario Metrics:

```
scenario_executions_total{scenario_name, status}
scenario_events_injected_total{scenario_name}
```

Redis Caching Strategies

Cache Patterns Used

1. Cache-Aside (Lazy Loading): Device configurations, user profiles **2. Write-Through:** Device state (must always be current) **3. Pub/Sub:** Real-time UI updates, agent responses

Redis Key Structure

```
# Device state (Hash)
device:state:{device_id}
  - state: "locked"
  - battery: 85
  - signal: -45
  - updated_at: 1705329600

# Device online status (String with TTL)
```

```
device:online:{device_id} = "1"
  TTL: 60 seconds

# User's devices (Set)
user:devices:{user_id} = [device_id1, device_id2, ...]

# Agent conversation cache (for quick context)
agent:context:{user_id} = "{current_mode, recent_actions, ...}"
  TTL: 1 hour

# Rate limiting
ratelimit:user:{user_id}:minute = count
  TTL: 60 seconds
```

---

n8n Workflows

Workflow 1: Motion Alert at Night

Trigger: Webhook from Event Processor **Flow:** Check time → Check arm state → Check agent recommendation → Send notification

Workflow 2: Device Offline Escalation

Trigger: Webhook when heartbeat missed **Flow:** Wait 2 min → Check if back online → If not, notify user → Wait 10 min → Escalate to admin

Workflow 3: Daily Summary Report

Trigger: Cron (9:00 AM daily) **Flow:** Call Analytics API → Generate summary → Send email

Workflow 4: Agent-Triggered Actions

Trigger: Webhook from Agentic AI Service **Flow:** Execute complex multi-step automations that agent orchestrates

---

UI Screens

1. Login Screen

```
┌─────────────────────────────────────┐
│         🏠 HomeGuard IoT              │
│                                     │
│   Select Demo User:                 │
│   ┌─────────────────────────────┐   │
│   │ ▼ john@demo.com             │   │
│   └─────────────────────────────┘   │
```

```
│                              │
│  Password: [demo123     ]         │
│                              │
│  [ Login ]                       │
└──────────────────────────────────────────────┘
```

2. Dashboard with AI Chat

```
┌──────────────────────────────────────────────────────────────────────────┐
│ 🏠 HomeGuard                        [John Smith ▼] [Logout] │
├──────────────────────────────────────────────────────────────────────────┤
│                                          │
│  ┌──────────────────────────────────────────────┐
┌──────────────────────────────┐ │
│ │ System Status: ● All Systems Online   │ │ 💬 HomeGuard AI       │ │
│ │                          │ │                 │ │
│ │ ┌────────┐ ┌────────┐ ┌────────┐ ┌────────┐ │ │ Agent: Good morning! Your │ │
│ │ │ 🚪 Front │ │ 👁️ Motion │ │ 🌡️ Therm │ │ 🚗 Garage │ │ │ home is secure. Front   │ │
│ │ │ Door │ │ Sensor │ │    │ │ Door │ │ │ door lock battery at 15% │ │
│ │ │ 🔒 Locked │ │ ● Active │ │ 72°F │ │ ▼Closed │ │ │ - replace soon.      │ │
│ │ │ 2m ago │ │Just now │ │Set:70° │ │ 1h ago │ │ │               │ │
│ │ └────────┘ └────────┘ └────────┘ └────────┘ │ │ You: I'm leaving for    │ │
│ │                          │ │ vacation tomorrow      │ │
│ │ 📊 Live Events          [View All] │ │                │ │
│ │ ├─ 14:32:05 Motion - Living Room     │ │ Agent: I'll prepare your │ │
│ │ ├─ 14:31:42 Thermostat - 72°F      │ │ home! Setting:       │ │
│ │ ├─ 14:30:15 Front door unlocked     │ │ ✅ Random light schedule │ │
│ │ └─ 14:28:33 Motion - Front Door     │ │ ✅ HVAC to 78°F      │ │
│ │                          │ │ ✅ Enhanced alerts     │ │
│ └──────────────────────────────────────────────┘ │                 │ │
│                          │ [Type message...]  [Send] │ │
│ 🎬 Demo Scenarios               └──────────────────────────────┘ │
│
├──────────────────────────────────────────────────────────────────────────┤
│
│ │ [🏖️ Vacation] [🎉 Party] [🚨 Break-in] [🔋 Battery] [📦 Package] [🐱 Pet] │ │
│ │ [🔥 Fire] [👵 Wellness] [⚡ Energy] [📅 Day-in-Life] [Custom...]     │ │
│
└──────────────────────────────────────────────────────────────────────────┤
│
│                                          │
│ 🔗 [📊 Grafana] [⚡ n8n Workflows] [📈 Analytics] [🔧 System Health]     │
```

```
│                                      │
└───────────────────────────────────────────────────────────────────────────────────
```

3. Demo Scenario Panel (Expanded)

```
┌─────────────────────────────────────────────────────────────────────────┐
│ 🎬 Demo Scenarios                    [Admin Only]   │
├─────────────────────────────────────────────────────────────────────────┤
│                                     │
│ Quick Triggers:                          │
│ ┌───────────────────────────────────────────────────────────────┐  │
│ │ [🏖 Vacation Mode] [🎉 Party Mode]  [🚨 Break-in]  │  │
│ │ [🔋 Low Battery]  [📦 Package]    [🔥 Fire]       │  │
│ │ [🐱 Pet Motion]   [👵 Wellness]   [🚗 Unknown Car] │  │
│ │ [⚡ Energy Insight] [📅 Day-in-Life]  [🔓 Guest]     │  │
│ └───────────────────────────────────────────────────────────────┘  │
│                                     │
│ Or type natural language to agent:                  │
│ ┌───────────────────────────────────────────────────────────────┐  │
│ │ "Simulate: Mom arriving Thursday at 2pm"          │  │
│ └───────────────────────────────────────────────────────────────┘  │
│                                     │
│ Timeline Playback:                          │
│ [▶ Play "Day in the Life" - 24hr compressed to 5min]        │
│ [▶ Play "Break-in Scenario" - Full incident response]      │
│                                     │
│ Currently Running: 🟢 break_in (T+5s of 30s)            │
│ [⏹ Stop]                                 │
│                                     │
└─────────────────────────────────────────────────────────────────────────┘
```

Deployment (Rancher/K8s)

Namespace Structure

```yaml
```

```yaml
namespaces:
  - homeguard-apps        # Microservices
  - homeguard-data        # Databases
  - homeguard-messaging   # Kafka
  - homeguard-ai          # AI services
  - homeguard-observability  # Grafana, Prometheus, Loki
  - homeguard-automation  # n8n
```

Helm Charts / Operators

Component	Deployment Method
PostgreSQL + TimescaleDB	Bitnami Helm chart
ScyllaDB	Scylla Operator
MongoDB	Bitnami Helm chart
Redis	Bitnami Helm chart
Kafka	Strimzi Operator
Grafana	Grafana Helm chart
Prometheus	kube-prometheus-stack
Loki	Grafana Loki Helm chart
n8n	n8n Helm chart

Resource Estimates

Component	CPU Request	Memory Request
PostgreSQL + TimescaleDB	500m	1Gi
ScyllaDB	1000m	2Gi
MongoDB	500m	1Gi
Redis	250m	512Mi
Kafka + Zookeeper	1000m	2Gi
Grafana	250m	256Mi
Prometheus	500m	1Gi
Loki	250m	256Mi
n8n	250m	256Mi
Go Services (7x)	100m each	128Mi each
Python Services (3x)	250m each	512Mi each
React UI	50m	64Mi

Component	CPU Request	Memory Request
Total	~6 CPU	~12Gi

Project Milestones

Phase 1: Infrastructure

- [ ] Rancher/K8s cluster setup
- [ ] Deploy all databases (PostgreSQL, TimescaleDB, ScyllaDB, MongoDB, Redis)
- [ ] Deploy Kafka
- [ ] Deploy observability stack (Grafana, Prometheus, Loki)
- [ ] Deploy n8n

Phase 2: Core Services

- [ ] API Gateway with rate limiting, circuit breaker
- [ ] User Service + seed data
- [ ] Device Service + seed data
- [ ] Device Ingest Service
- [ ] Event Processor

Phase 3: Real-Time Features

- [ ] Notification Service (WebSocket)
- [ ] Device Simulator
- [ ] Redis pub/sub integration
- [ ] Real-time UI updates

Phase 4: Analytics & Automation

- [ ] Analytics Service
- [ ] TimescaleDB continuous aggregates
- [ ] n8n workflows
- [ ] Automation rules engine

Phase 5: AI/ML Services

- [ ] Anomaly ML Service (scikit-learn)
- [ ] MCP Server
- [ ] Agentic AI Service (Gemini integration)
- [ ] Agent conversation memory (MongoDB)

Phase 6: Demo System

- [ ] Scenario Engine

- [ ] Pre-built scenarios (12 scenarios)
- [ ] Demo control panel in UI
- [ ] Chat interface for AI agent

Phase 7: UI

- [ ] Login screen
- [ ] Dashboard with live updates
- [ ] AI Chat panel
- [ ] Demo scenario panel
- [ ] Device detail view
- [ ] Events feed
- [ ] Automations management
- [ ] Analytics charts
- [ ] Admin system view

Phase 8: Polish

- [ ] Grafana dashboards
- [ ] Documentation
- [ ] Demo script
- [ ] Performance tuning

Interview Talking Points

Architecture Decisions

Q: Why multiple databases?

> "Each database is optimized for its workload. PostgreSQL for ACID transactions on user data, ScyllaDB for high-throughput event ingestion, TimescaleDB for time-series analytics with automatic partitioning and compression, MongoDB for flexible device configurations and AI agent memory, and Redis for sub-millisecond current state queries."

Q: How does it handle failures?

> "Circuit breakers prevent cascade failures - if MongoDB is slow, we trip the circuit and return cached data from Redis. Rate limiting protects against traffic spikes. All Kafka consumers are idempotent so we can replay events safely. The AI agent gracefully degrades if Gemini API is unavailable."

Q: How would this scale?

> "Horizontally at every layer. Kafka partitions can increase from 3 to 30. ScyllaDB adds nodes linearly. Redis can cluster. Go services are stateless - just add replicas. The simulator can pump 10,000 events/sec to prove it."

Q: Why Kafka instead of direct writes?

> "Decoupling. The 1000 applications at my company write directly to SQL Server, causing bottlenecks. With Kafka, producers are fire-and-forget, consumers scale independently, and we can replay events for debugging or reprocessing."

AI/ML Deep Dives

Q: Why use an LLM for home automation?

> "Current systems are rule-based and reactive. The AI agent understands context - it knows that 2 AM motion when you're home and have a pet registered is probably not a threat. It reduces false alarms by 80% while catching real threats. It can also handle complex requests like 'I'm leaving for vacation' with a single conversation."

Q: How do you prevent the AI from making dangerous decisions?

> "Three safeguards: 1) Security-critical actions require confirmation, 2) The MCP server validates all commands before execution, 3) All actions are logged and can be audited. The AI explains its reasoning so users understand why it took an action."

Q: What happens if Gemini API is down?

> "Circuit breaker trips after 3 failures. The system falls back to rule-based automations stored in MongoDB. Users see a notification that AI features are temporarily limited. The system continues to function for basic operations."

Q: How does the anomaly detection work?

> "IsolationForest trained on 30 days of normal patterns. It learns what's typical for each device and user. Events that deviate significantly get flagged. The AI agent uses these anomaly scores as context when deciding whether to alert the user."

Demo Script

```
1. START: Show dashboard with live simulator running
   "Here's a typical smart home with devices generating real events..."

2. AI CHAT: Type "I'm leaving for vacation tomorrow for 2 weeks"
   "Watch how the AI understands context and creates a comprehensive plan..."
   → Show agent reasoning
   → Show actions being taken
   → User approves

3. SCENARIO: Click "Break-in Scenario"
   "Now let's see how the system handles a real threat..."
   → Real-time event flow on dashboard
   → Agent coordinating response
   → Incident documentation generated
```

```
4. FALSE ALARM: Click "Pet Motion at 2 AM"
   "Current systems would wake you up. Watch this..."
   → Agent reasons through context
   → Decides NOT to alert
   → Shows learning

5. SHOW BACKEND: Open Grafana
   "All these events flowed through Kafka, got scored by ML,
    and the agent made decisions in under 500ms..."
   → Show Kafka throughput
   → Show AI response times
   → Show anomaly detection metrics
```

Configuration

Environment Variables

```yaml
# Gemini API (User Provided)
GEMINI_API_KEY: "<user-provided-key>"

# Database connections
POSTGRES_URL: "postgresql://user:pass@postgres:5432/homeguard"
TIMESCALE_URL: "postgresql://user:pass@timescale:5432/analytics"
SCYLLA_HOSTS: "scylla-0.scylla,scylla-1.scylla,scylla-2.scylla"
MONGO_URL: "mongodb://mongo:27017/homeguard"
REDIS_URL: "redis://redis:6379"
KAFKA_BROKERS: "kafka-0.kafka:9092,kafka-1.kafka:9092"

# Service URLs
MCP_SERVER_URL: "http://mcp-server:8080"
ANOMALY_SERVICE_URL: "http://anomaly-ml:8080"
N8N_WEBHOOK_BASE: "http://n8n:5678/webhook"

# Feature flags
ENABLE_AI_AGENT: true
ENABLE_ANOMALY_DETECTION: true
ENABLE_DEMO_MODE: true
```

Document Version: 2.0 Project: HomeGuard IoT Platform Stack: Go, Python, React, PostgreSQL, TimescaleDB, ScyllaDB, MongoDB, Redis, Kafka, Grafana, Prometheus, Loki, n8n, Google Gemini

Claude Build Prompt

Use the following prompt to instruct Claude to build this project:

PROMPT FOR CLAUDE

I want you to help me build the HomeGuard IoT Platform - a production-grade portfolio project demonstrating polyglot persistence, event-driven architecture, and Agentic AI for smart home security.

PROJECT OVERVIEW

This is a complete IoT platform deployed on Rancher (local Kubernetes) that showcases:
- **Polyglot Persistence**: PostgreSQL, TimescaleDB, ScyllaDB, MongoDB, Redis, Kafka
- **Event-Driven Architecture**: Kafka for event streaming, Redis pub/sub for real-time
- **Agentic AI**: Google Gemini-powered intelligent assistant with MCP (Model Context Protocol)
- **ML/Anomaly Detection**: scikit-learn for pattern detection and predictive maintenance
- **Full Observability**: Grafana, Prometheus, Loki
- **Demo System**: Scenario engine for triggering pre-built demo scenarios

TECHNOLOGY STACK

Languages:
- Go: All backend services (API Gateway, User Service, Device Service, Device Ingest, Event Processor, Notification Service, Scenario Engine)
- Python: Analytics Service, Anomaly ML Service, Agentic AI Service, MCP Server
- TypeScript/React: Frontend UI with AI Chat interface

Databases:
- PostgreSQL: User accounts, auth (relational, ACID)
- TimescaleDB: Time-series sensor data (PostgreSQL extension with hypertables)
- ScyllaDB: High-volume IoT events (wide-column, write-heavy)
- MongoDB: Device configs, automation rules, agent memory (document store)
- Redis: Device state cache, pub/sub, rate limiting
- Kafka: Event streaming bus

AI/ML:
- Google Gemini API: LLM for agentic reasoning (I will provide the API key)
- scikit-learn: IsolationForest for anomaly detection, LinearRegression for predictions
- MCP Server: Exposes tools for Gemini to query/control the system

Observability:
- Grafana: Dashboards, alerting
- Prometheus: Metrics
- Loki: Log aggregation

Automation:
- n8n: Workflow automation

MICROSERVICES TO BUILD

1. **API Gateway (Go)**: Rate limiting, circuit breaker, JWT auth, routing
2. **User Service (Go)**: Auth, pre-seeded demo users (john@demo.com, sarah@demo.com, admin@demo.com)
3. **Device Service (Go)**: Device CRUD, configs (MongoDB), state (Redis)
4. **Device Ingest Service (Go)**: Kafka producer for device events
5. **Event Processor (Go)**: Kafka consumer → ScyllaDB, Redis, triggers n8n
6. **Notification Service (Go)**: WebSocket connections, Redis pub/sub → UI
7. **Analytics Service (Python)**: TimescaleDB queries, aggregations
8. **Device Simulator (Go)**: Generates realistic IoT events → Kafka
9. **Scenario Engine (Go)**: Executes demo scenarios, injects events into Kafka
10. **Anomaly ML Service (Python)**: IsolationForest anomaly detection, battery predictions
11. **Agentic AI Service (Python)**: Gemini integration, conversation handling, MCP client
12. **MCP Server (Python)**: Tools for LLM to query devices, events, control system

AI AGENT CAPABILITIES

The Agentic AI Service should:
- Understand natural language requests ("I'm leaving for vacation tomorrow")
- Use MCP tools to get home state, device info, events, anomalies
- Make autonomous decisions with context awareness
- Execute multi-step actions (vacation mode sets 5+ things at once)
- Learn user preferences and patterns
- Prevent false alarms by understanding context (2 AM motion + pet + no door breach = probably pet)
- Coordinate emergency responses (fire: unlock doors, kill HVAC, notify contacts)

DEMO SCENARIOS TO BUILD

Pre-built scenarios that can be triggered from the UI:
1. vacation_mode - User announces vacation, agent creates comprehensive plan
2. break_in - Glass break → motion → door sequence, full incident response
3. false_alarm_pet - 2 AM motion that agent correctly suppresses
4. guest_arrival - Expected guest with auto-unlock
5. low_battery - Battery prediction and notification
6. unusual_pattern - Door unlocks with no follow-up motion
7. party_mode - Adjusted sensitivity for party

8. fire_emergency - Full emergency coordination
9. package_delivery - Doorbell + package detection
10. wellness_check - No activity escalation
11. energy_insight - Pattern-based energy recommendations
12. day_in_life - 24hr compressed to 5min showing agent learning

UI REQUIREMENTS

React frontend with:
- Dashboard showing device status (real-time updates via WebSocket)
- AI Chat panel for conversing with agent
- Demo Scenario panel with quick trigger buttons
- Events feed (live scrolling)
- Device detail views
- Analytics charts (from TimescaleDB)
- Admin system health view

RESILIENCE PATTERNS

Implement in all services:
- Circuit breakers (sony/gobreaker for Go)
- Rate limiting (Redis-backed token bucket)
- Structured logging (JSON to Loki)
- Prometheus metrics
- Retry with exponential backoff
- Graceful degradation (AI falls back to rules if Gemini down)

DEPLOYMENT

All services deploy to Rancher/K8s with:
- Helm charts for databases
- Strimzi for Kafka
- ConfigMaps for configuration
- Secrets for API keys
- Resource limits

GETTING STARTED

Please help me build this step by step:
1. First, set up the infrastructure (databases, Kafka, observability)
2. Then build core services (API Gateway, User, Device, Event Processor)
3. Add real-time features (Notification Service, Device Simulator)
4. Build AI/ML services (Anomaly ML, MCP Server, Agentic AI Service)
5. Create Scenario Engine and demo scenarios

6. Build the React UI with chat interface
7. Create Grafana dashboards
8. Write deployment manifests

I have the complete requirements document with schemas, API endpoints, and implementation details. Let's start with [SPECIFIC COMPONENT].

Note: I will provide the Gemini API key as an environment variable (GEMINI_API_KEY).

END OF REQUIREMENTS DOCUMENT