**Design Document: Event Processing**

**1. Introduction**

**a. Purpose and Scope**

This document outlines the design for processing events with 'Item Type = "Place Holder"'. The implementation will fetch relevant item details based on

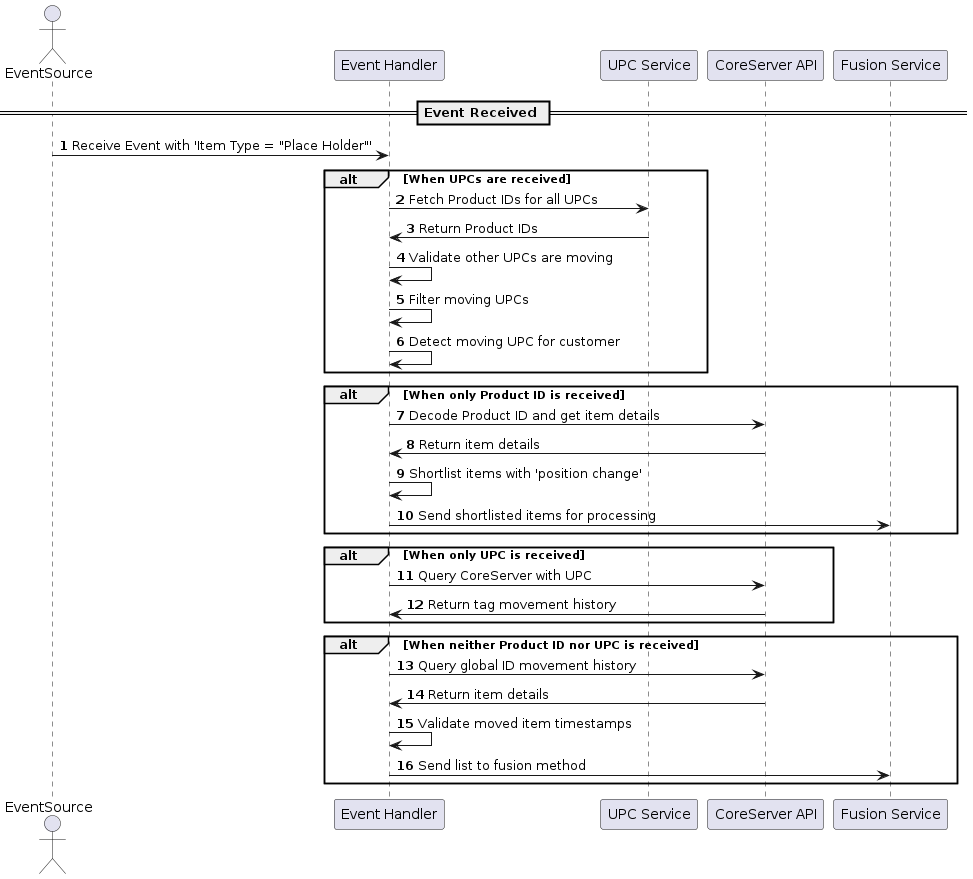
different input conditions, ensuring proper data flow, validation, and integration with external services.

**b. Problem Statement**

The system needs to process incoming events that contain UPCs or Product IDs to detect if a customer has picked up any product. The challenge lies in correctly identifying

moving UPCs, validating item movements, and ensuring efficient querying of external services like SmartLens CoreServer API and Fusion Service.

**2. Flow Diagram**



---

**3. Description of New Features**

**UPC Bunch Handling**: Fetches product IDs for all UPCs from a service call.

**Moving UPC Filtering**: Validates and filters out only moving UPCs.

**Product ID Processing**: Decodes product ID and retrieves item details via SmartLens CoreServer API.

**Tag Movement History Query**: Fetches historical movement data when only UPC is available.

**Global ID Movement Query**: Retrieves item details based on global ID movement in the absence of Product ID and UPC.

**4. Component Responsibilities**

**a. Event Processor**

Receives and parses incoming events.

Determines if UPCs, Product IDs, or both are present.

Routes data for further processing.

**b. UPC Service**

- Fetches Product IDs for given UPCs.

- Validates and filters moving UPCs.

**c. SmartLens CoreServer API**

- Decodes Product IDs and fetches item details.

- Provides tag movement history for UPC queries.

- Retrieves details for globally moved items when Product ID and UPC are absent.

**d. Fusion Service**

- Processes shortlisted items based on position change events.

- Handles further processing and analytics.

---

**5.Service Request and Response**

# **Smart Lens Core API Documentation**

## **Overview**

The Smart Lens Core API provides location and status details of an item based on its unique Item ID. This API enables users to fetch real-time positional data, confidence scores, product identifiers, and status information for a given item.

## **Endpoint**

**GET** /api/item/{itemId}

## **Request**

### **Path Parameters**

| **Parameter** | **Type** | **Required** | **Description** |
| --- | --- | --- | --- |
| itemId | String | Yes | The unique identifier of the item to query |

## **Response**

### **Success Response (200 OK)**

{

"id": "30340789004D0D8000000074",

"x": 900,

"y": 900,

"z": 900,

"confidence": 60,

"floor": 0,

"type": "ITEM",

"timestamp": "2020-08-01T03:11:09Z",

"region": "4d03cf0b-f856-49f6-87fb-f8a40a10271d",

"site": "936c1954-dce8-4783-90d3-cb9a7db19698",

"productId": "00123456789029",

"idFormat": "EPC",

"associatedId": "",

"associatedIdFormat": "",

"state": "MISSING",

"reason": "NONE"

}

### **Response Fields**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Unique identifier of the item |
| x | Integer | X-coordinate of the item location |
| y | Integer | Y-coordinate of the item location |
| z | Integer | Z-coordinate of the item location |
| confidence | Integer | Confidence level (0-100) of the item's location |
| floor | Integer | Floor number where the item is located |
| type | String | Type of the entity (e.g., ITEM) |
| timestamp | String | Timestamp of the last recorded location in ISO 8601 format |
| region | String | Region identifier where the item is located |
| site | String | Site identifier of the item's location |
| productId | String | Associated product identifier |
| idFormat | String | Format of the item identifier (e.g., EPC) |
| associatedId | String | Linked identifier, if available |
| associatedIdFormat | String | Format of the associated ID |
| state | String | Current status of the item (e.g., MISSING) |
| reason | String | Additional details for the state |

**6. Data Flow and Dependencies**

- The Event Processor receives data and routes it to the appropriate services.

- The UPC Service fetches and filters moving UPCs.

- The SmartLens CoreServer API fetches details based on Product IDs or tag movement history.

- The Fusion Service processes shortlisted items.

**7. Error Handling and Edge Cases**

**a. Incomplete Data Handling**

- If UPCs are missing, the system falls back to querying Global ID movement.

- If Product ID is missing, the system queries UPC movement history.

**b. Inconsistent Data Handling**

- Validates timestamps to ensure recent movement data is considered.

- Ensures API response consistency before processing.

**c. Erroneous Data Handling**

- Implements retries for failed API calls.

- Logs errors for missing or malformed data.

---

**8. Performance Considerations**

**a. Performance Benchmarks**

- API response time: SmartLens CoreServer API calls should not exceed 500ms.

- Data processing time: UPC filtering and movement validation should be completed within 200ms.

- Storage efficiency: Cosmos DB should be optimized for querying high-frequency movement events.

**b. Optimization Techniques**

- Batch processing for UPC lookups to reduce API calls.

- Caching frequently accessed product movement data.

- Asynchronous API requests to improve throughput.

---

**Conclusion**

This design ensures efficient event processing for 'Place Holder' items, leveraging external services for data retrieval and movement validation. By structuring components with clear responsibilities, defining data formats, and planning for error handling, the system can efficiently process product movement events with minimal latency.

**Pseudocode**

class EventProcessor:

def \_\_init\_\_(self, upc\_service, core\_server, fusion\_service):

self.upc\_service = upc\_service

self.core\_server = core\_server

self.fusion\_service = fusion\_service

def process\_event(self, event):

if 'upcs' in event:

product\_ids = self.upc\_service.fetch\_product\_ids(event['upcs'])

moving\_upcs = self.upc\_service.filter\_moving\_upcs(product\_ids)

self.fusion\_service.process\_items(moving\_upcs)

elif 'product\_id' in event:

item\_details = self.core\_server.fetch\_item\_details(event['product\_id'])

shortlisted\_items = self.core\_server.filter\_position\_changes(item\_details)

self.fusion\_service.process\_items(shortlisted\_items)

else:

moved\_items = self.core\_server.query\_global\_movement()

valid\_items = self.core\_server.validate\_movement\_timestamp(moved\_items)

self.fusion\_service.process\_items(valid\_items)

class UPCService:

def fetch\_product\_ids(self, upcs):

# Call external service to fetch product IDs

pass

def filter\_moving\_upcs(self, product\_ids):

# Validate and filter moving UPCs

pass

class CoreServer:

def fetch\_item\_details(self, product\_id):

# Call SmartLens CoreServer API to get item details

pass

def filter\_position\_changes(self, items):

# Shortlist items based on position change events

pass

def query\_global\_movement(self):

# Query global movement details

pass

def validate\_movement\_timestamp(self, items):

# Validate timestamps of moved items

pass

class FusionService:

def process\_items(self, items):

# Pass the list of items to fusion service for further processing

pass

# Example Usage

event\_processor = EventProcessor(UPCService(), CoreServer(), FusionService())

event = { "upcs": ["123456", "789012"] }

event\_processor.process\_event(event)