**System Overview**

The Payment Receipt Generation System comprises five core microservices:

1. **Receipt Generation Service**

* **Role**: Receives a transaction ID from the client, gathers necessary data, generates a receipt in JSON format, and triggers storage and analysis tasks.
* **Interactions**: This service communicates with the **User Info Service** and **Payment Info Service** to retrieve data, and then with the **Receipt Storage Service** and **Receipt Analysis Service** for further processing.

1. User Info Service

* **Role**: Provides user-related data (e.g., name, contact information) associated with a specific transaction ID.
* **Interactions**: Responds to data requests from the **Receipt Generation Service** to deliver user information synchronously.

1. Payment Info Service

* **Role**: Manages and provides payment-related data, such as amount, payment method, and transaction status, based on the transaction ID.
* **Interactions**: Responds to data requests from the **Receipt Generation Service** to deliver payment information synchronously.

1. Receipt Storage Service

* **Role**: Stores the generated receipt data for record-keeping and future reference.
* **Interactions**: Receives the receipt data from the **Receipt Generation Service** for storage asynchronously to avoid blocking further processing.

1. Receipt Analysis Service

* **Role**: Analyzes the receipt for insights or reporting purposes.
* **Interactions**: Receives the receipt data from the **Receipt Generation Service** asynchronously for non-blocking, parallel analysis.

A diagram of a receipt

Description automatically generated

**(High level Diagram of System)**

Interaction Flow

1. **Client to Receipt Generation Service**:

* **Type**: Synchronous (REST API)
* **Reasoning**: Using a REST API allows a standard HTTP-based communication, which is often simpler and widely compatible for client applications. REST is stateless, making it suitable for handling individual receipt requests from clients without maintaining session state on the server. It also offers flexibility in terms of request handling, caching, and error management.

1. **Receipt Generation Service to User Info and Payment Info Services**:

* **Type**: Synchronous (gRPC)
* **Reasoning**: The **Receipt Generation Service** requires data from both **User Info** and **Payment Info Services** to generate the receipt accurately. Since both pieces of data are essential for receipt creation, synchronous communication is appropriate here to retrieve data promptly within the receipt generation flow.

1. **Receipt Generation Service to Receipt Storage Service**:

* **Type**: Asynchronous (Kafka)
* **Reasoning**: Once the receipt is generated, the **Receipt Generation Service** sends the JSON data to the **Receipt Storage Service** using Kafka, an asynchronous communication method. This setup ensures the **Receipt Generation Service** can quickly offload storage tasks, freeing it to handle other requests without delay.

1. **Receipt Generation Service to Receipt Analysis Service**:

* **Type**: Asynchronous (Kafka)
* **Reasoning**: The **Receipt Analysis Service** performs analytics on receipt data, which isn’t required immediately by the client. Kafka enables the **Receipt Generation Service** to dispatch receipt data to the **Receipt Analysis Service** asynchronously, ensuring that analysis happens in parallel without impacting the receipt generation flow.

**Summary of Synchronous and Asynchronous Interactions:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Interaction** | **Protocol** | **Type** | **Explanation** |
| Client → Receipt Generation Service | REST | Synchronous | REST API enables simple HTTP-based, stateless communication with the client |
| Receipt Generation → User Info Service | gRPC | Synchronous | |  | | --- | |  |  |  | | --- | | gRPC for quick retrieval of necessary user data | |
| Receipt Generation → Payment Info Service | gRPC | Synchronous | gRPC for efficient payment data retrieval |
| Receipt Generation → Receipt Storage | Kafka | Asynchronous | Offloads storage tasks to prevent blocking receipt generation |
| Receipt Generation → Receipt Analysis | Kafka | Asynchronous | Allows parallel analysis processing without impacting generation performance |

**Benefits of This Design:**

1. **Decoupling and Scalability**: Kafka-based asynchronous calls ensure that **Receipt Storage** and **Receipt Analysis Services** can operate independently, scaling without impacting synchronous services.
2. **Responsiveness**: Synchronous gRPC interactions are used only where real-time data retrieval is necessary, keeping client-facing interactions fast.
3. **Flexibility for Future Extensions**: The design supports future services (e.g., additional analysis tools or notifications) with minimal impact, especially with Kafka enabling easy event-driven expansion.