## **Solution Design Document Transaction Management System**

## **1. System/Feature Summary** The main objective of this System/Feature is to handle transactions from a wide range of different processors with ability to scale furthermore in the future. Processors have different schemas regarding the webhook integration. However, the system/feature should be able to expose a webhook endpoint for receiving a specific Processor request and forward it to gRPC service using an inter-service communication channel which will process the given request and return the appropriate response to the client.

### **2. Proposed Solutions**

#### **2.1 Solution 1: Monolithic Architecture**

* **Description**: A single service handling both REST and gRPC endpoints, with shared business logic and database access.
* **Pros**:
  + Simplicity in deployment and management.
  + Lower latency due to no inter-service communication.
* **Cons**:
  + Scalability limitations due to single service constraints.
  + Harder to manage and maintain as the project grows.

#### **2.2 Solution 2: Microservices Architecture**

* **Description**: Separate services for REST (Gateway Service) and gRPC (Transaction Service), each with distinct responsibilities.
* **Pros**:
  + Scalability: Services can be scaled independently.
  + Flexibility: Easier to maintain, update, and expand individual services.
* **Cons**:
  + Increased complexity in communication and deployment.
  + Requires more robust monitoring and logging solutions.

#### **2.3 Solution 3: Domain-Driven Design (DDD) with Bounded Contexts**

* **Description**: The architecture is structured around distinct bounded contexts that align with business domains. Each context is encapsulated and has well-defined interfaces for interaction with other contexts.
* **Pros**:
  + Clarity: Clear boundaries for domain logic and responsibilities
  + Decoupling: Reduces dependencies between different parts of the system
* **Cons**:
  + Complexity: Requires careful planning and understanding of business domains to define appropriate bounded contexts.
  + Potential overhead: Inter-context communication can introduce overhead, especially if not properly managed.

### **3. Chosen Solution**

**Chosen Solution: Microservices Architecture**The microservices architecture was selected for its scalability, flexibility, and maintainability advantages. This architecture allows independent development, deployment, and scaling of the Gateway and Transaction services. Despite added complexity, the architecture's ability to handle growing and evolving system requirements makes it the preferred solution.

### **4. Assumption and Prerequisites**

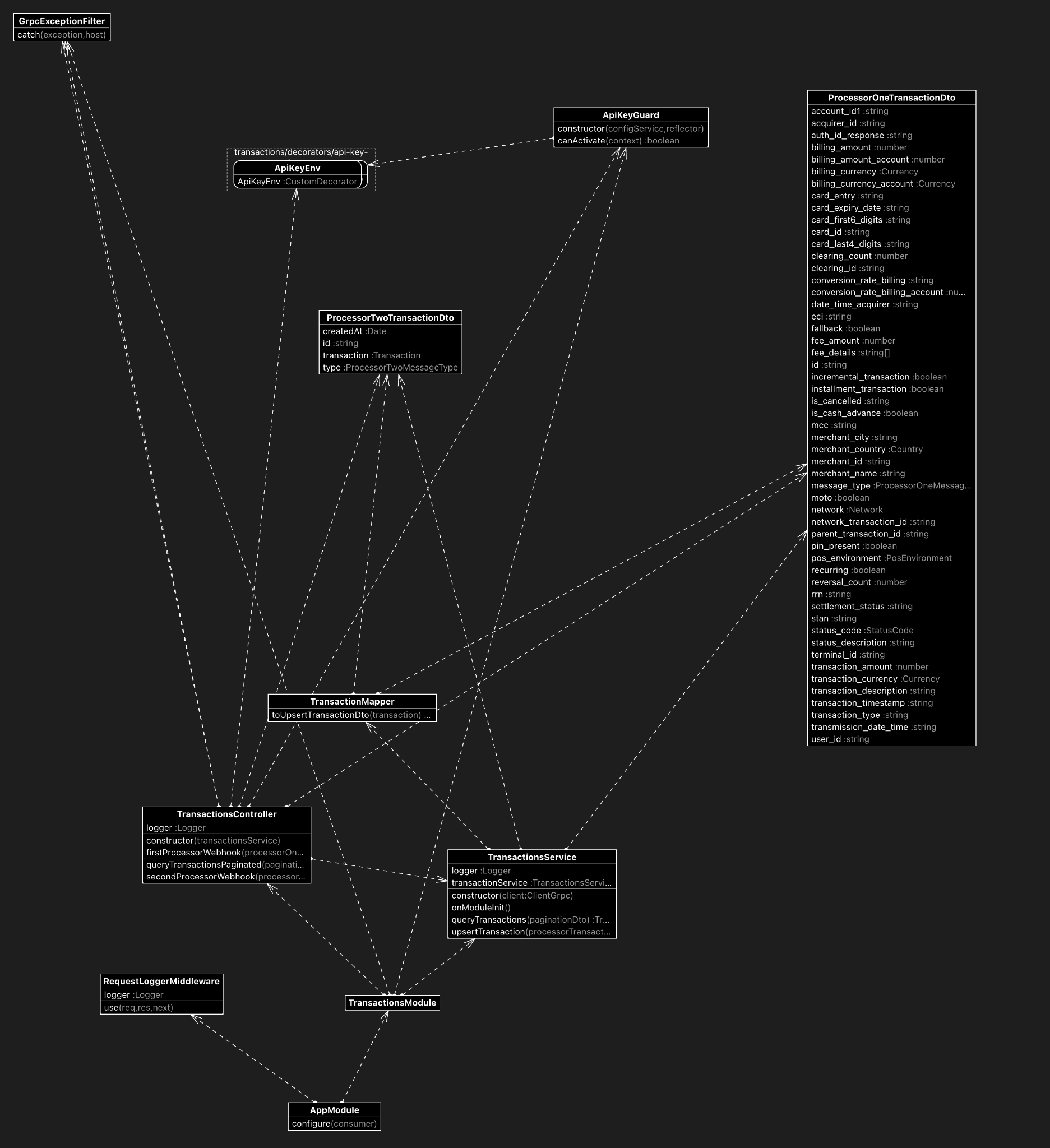
This solution is assuming the following givens:

* Accounts database table
* Cards database table

### **5. Diagrams for the Chosen Solution**

#### **5.1 Class Diagrams**

**5.1.1 Gateway class diagram**

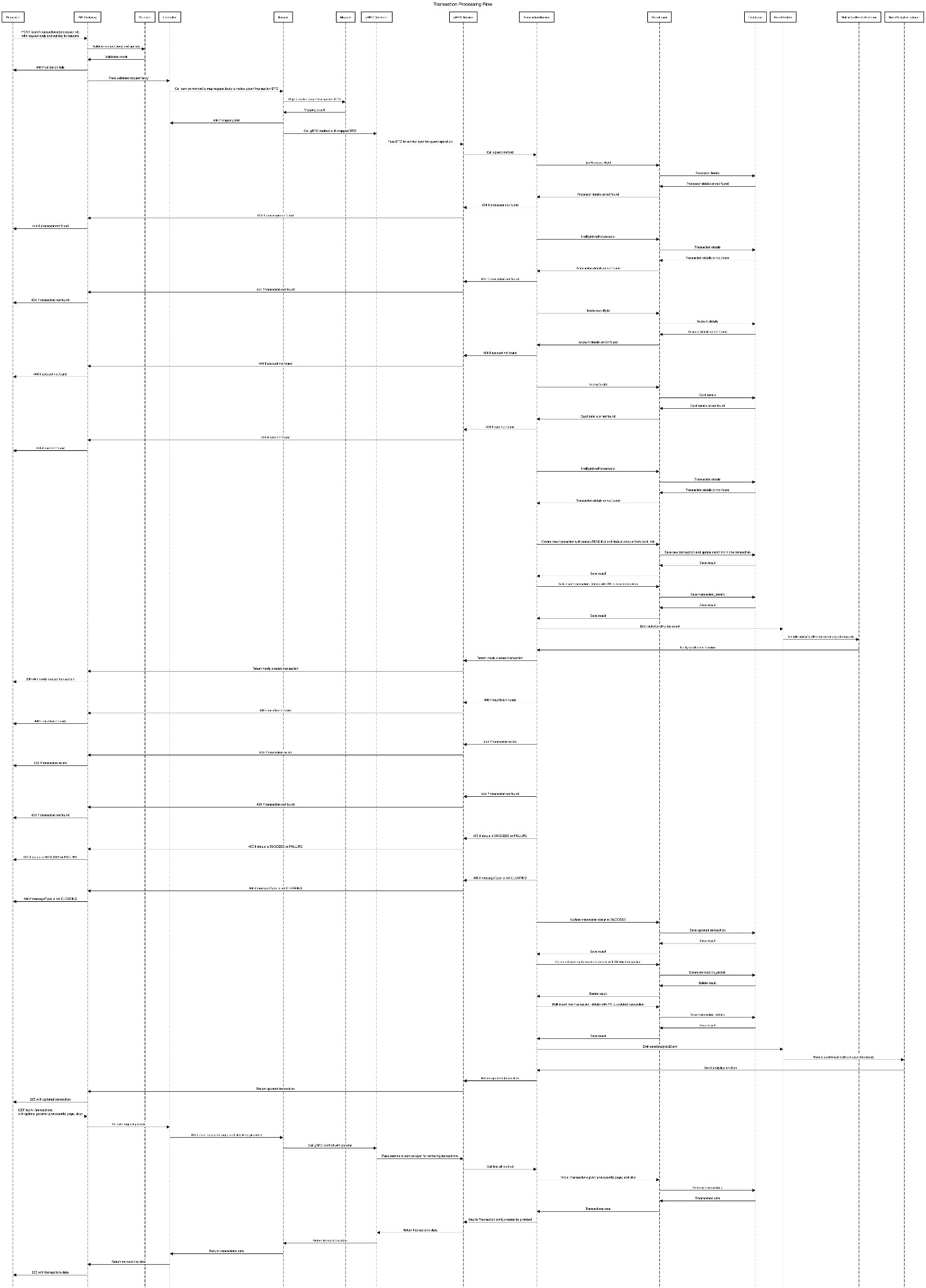


**5.1.2 Transactions class diagram**

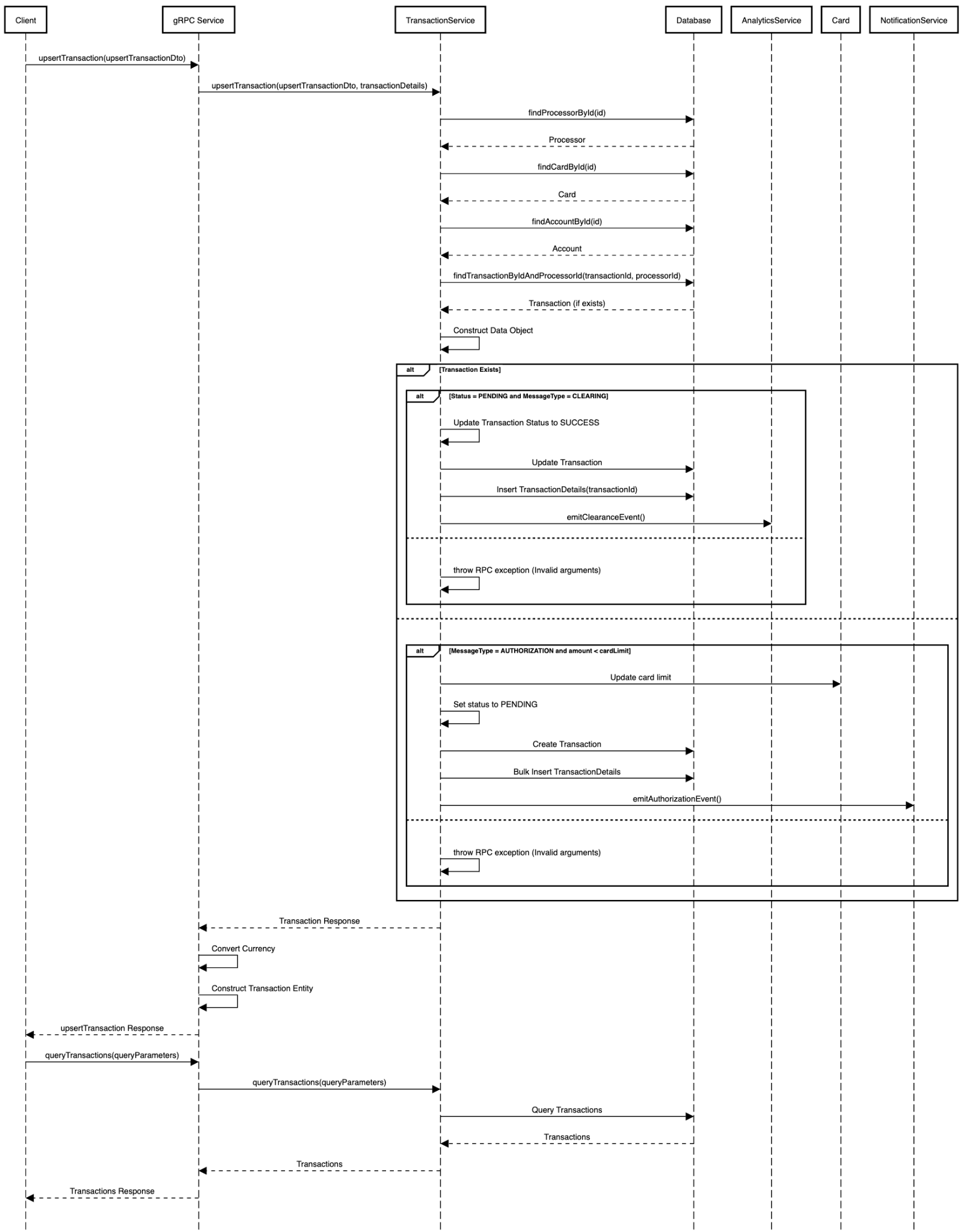
#### 

#### **5.2 Sequence Diagrams**

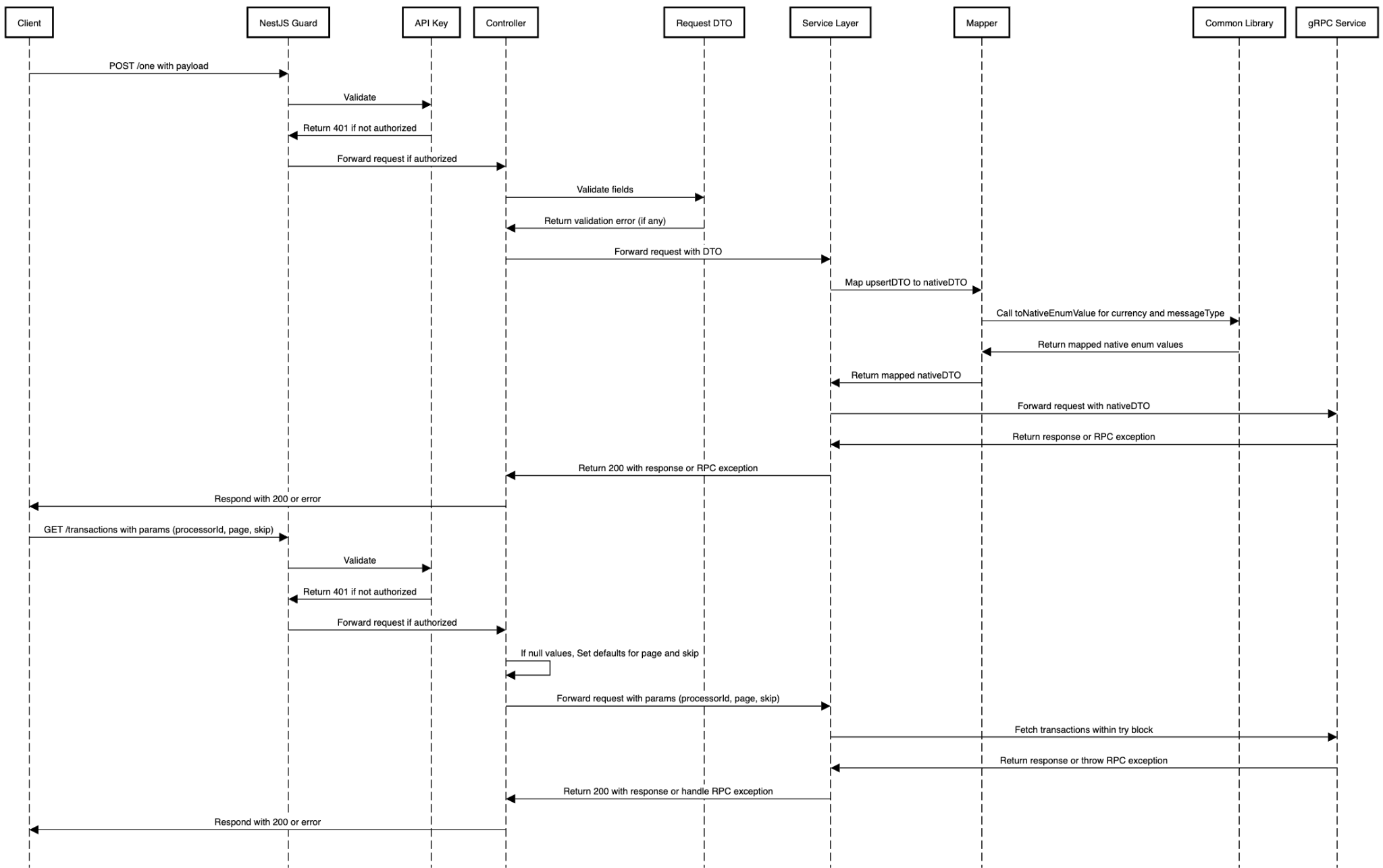
**5.2.1 Overall Sequence Diagram**

****

**5.2.2 Transactions Sequence Diagram**

****

**5.2.3 Gateway Sequence Diagram**

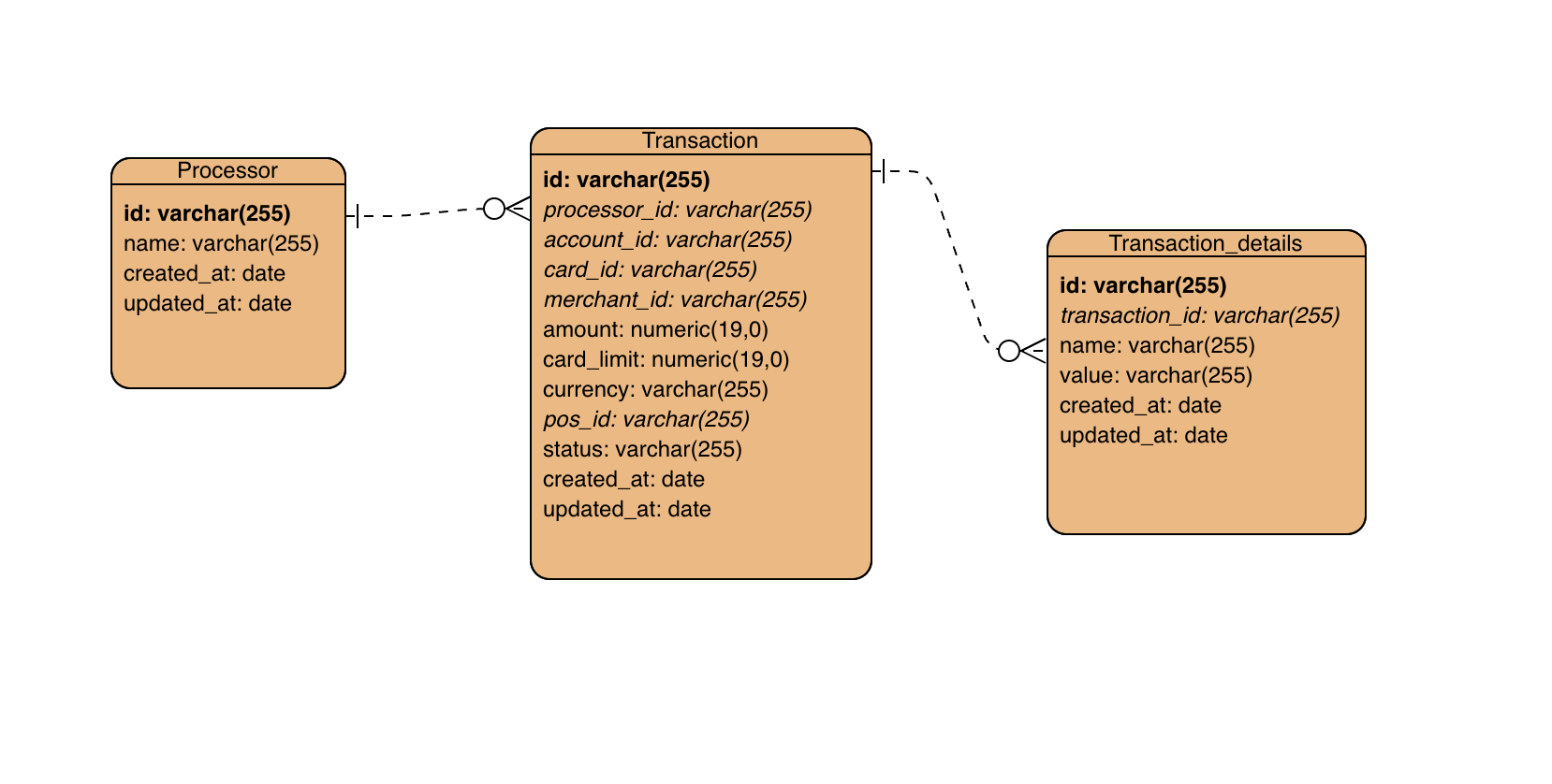
****

#### 

#### 

#### 

#### **5.3 Entity-Relationship Diagram (ERD)**



### 

### **6. Technical Description**

#### **6.1 Added API Endpoints**

**REST API (Gateway Service)**

* **POST /{one/two}**
  + **Description**: Receives webhook transactions from processors.
  + **Request Body**: Varies by processor, standardized via DTOs.
  + **Response**: Acknowledgement of receipt.
* **GET /transactions**
  + **Description**: Retrieves a list of transactions with pagination and filtering.
  + **Response**: Array of transactions with details.

**gRPC Services (Transaction Service)**

* **UpsertTransaction**
  + **Request**: NativeUpsertTransactionDTO
  + **Response**: NativeTransactionProtoEntity
* **QueryTransactions**
  + **Request**: { "processorId": string, "page": number, "limit": number }
  + **Response**: Array of transaction objects.