**Task 3 : Scalable Banking API**

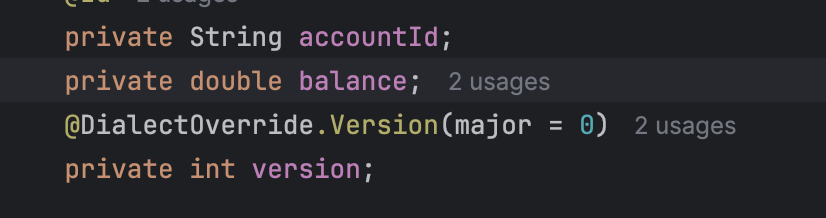
**Project Link :** [**Github**](https://github.com/vishvas04/Zeta_Assignment/tree/main/Scalable%20Banking%20API)

I built this API using **Java Spring Boot** andI used **Spring Web, Spring Data JPA** for database interactions and **H2** ( for testing )**, PostgreSQL Driver** and **lombok** ( for boiler plate codes).

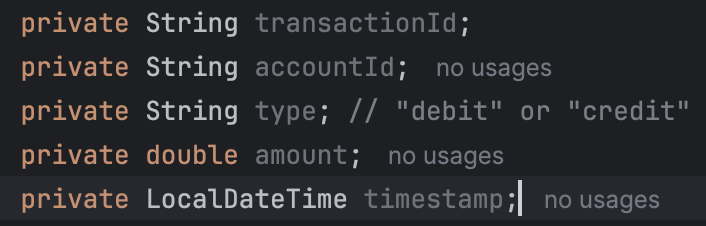
This API is built in such a way that it handles multiple transactions with a focus on atomicity, concurrency safety, and resilience. Built using Java Spring Boot, it ensures consistency during failures and scales for millions of daily transactions. It supports debit, credit, balance enquiry and account creation operations concurrently and ensures atomicity and clearly mentions the errors as well.

**Database Schema:**

Accounts:



Transactions:



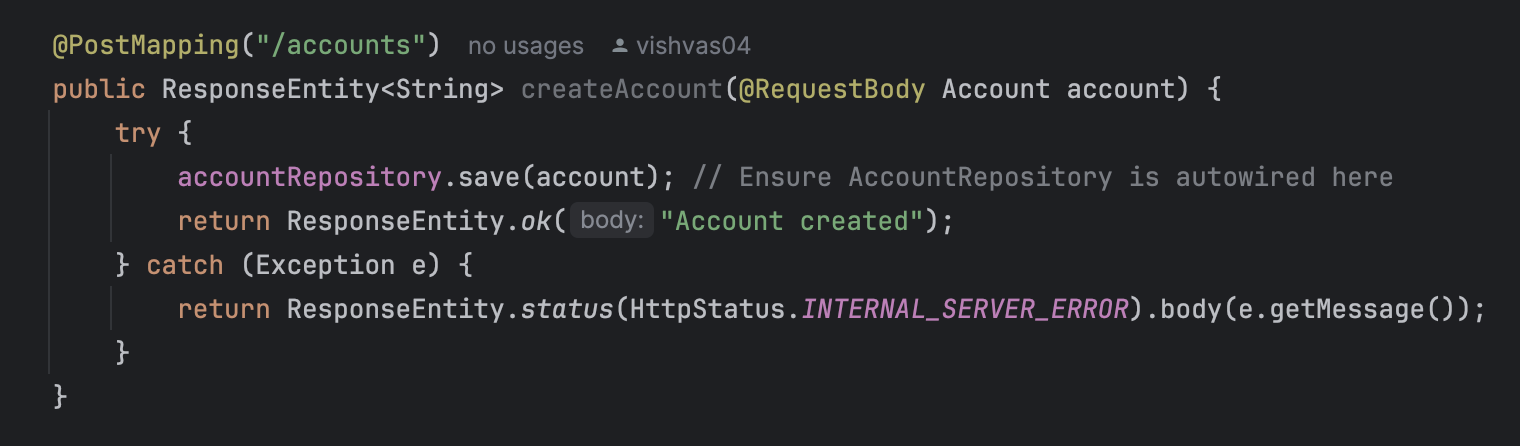
**Optimizations**

**Indexes**: accountId (both tables), timestamp (transactions).

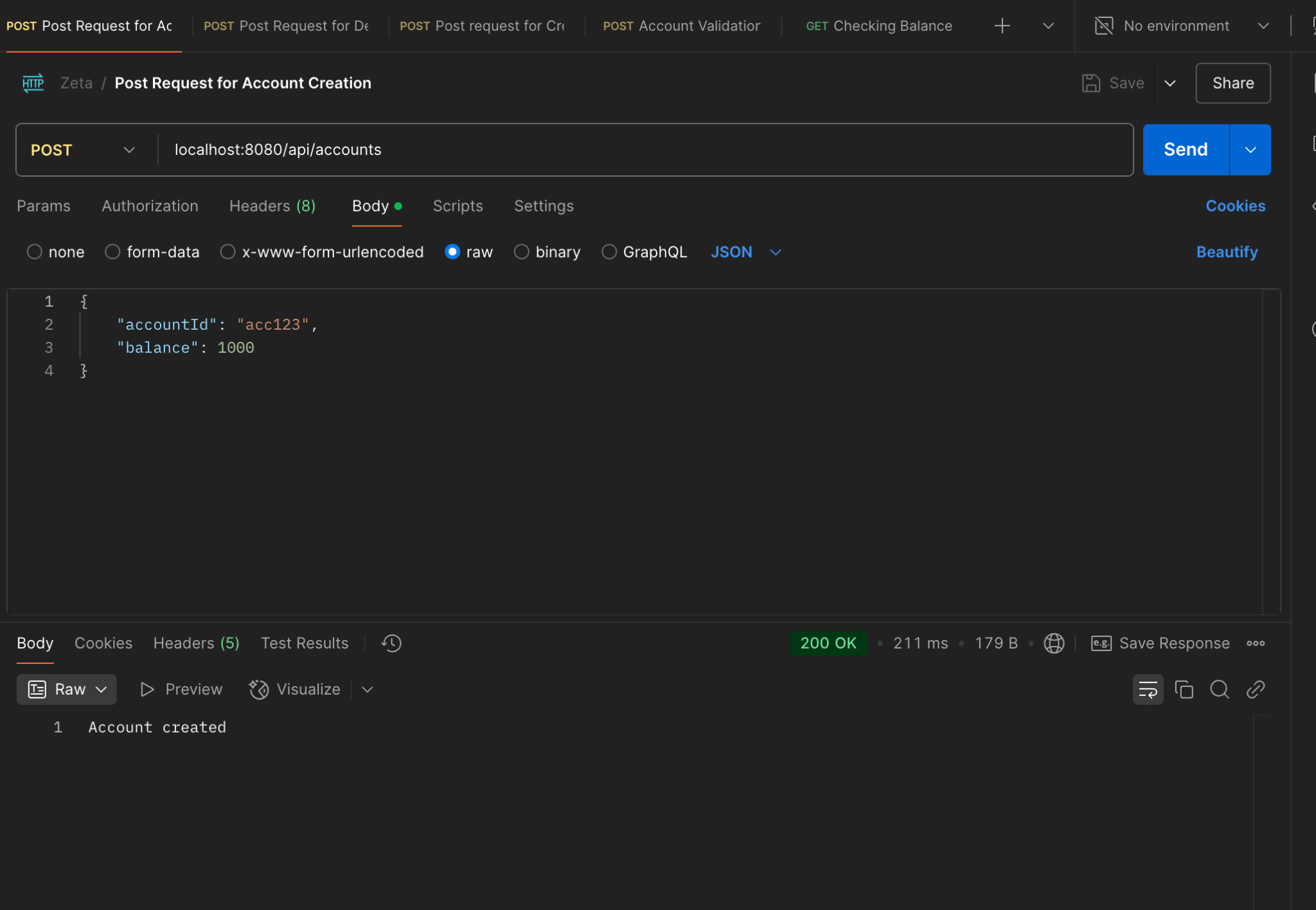
**Partitioning**: Split transactions by date for faster queries.

**API Design**

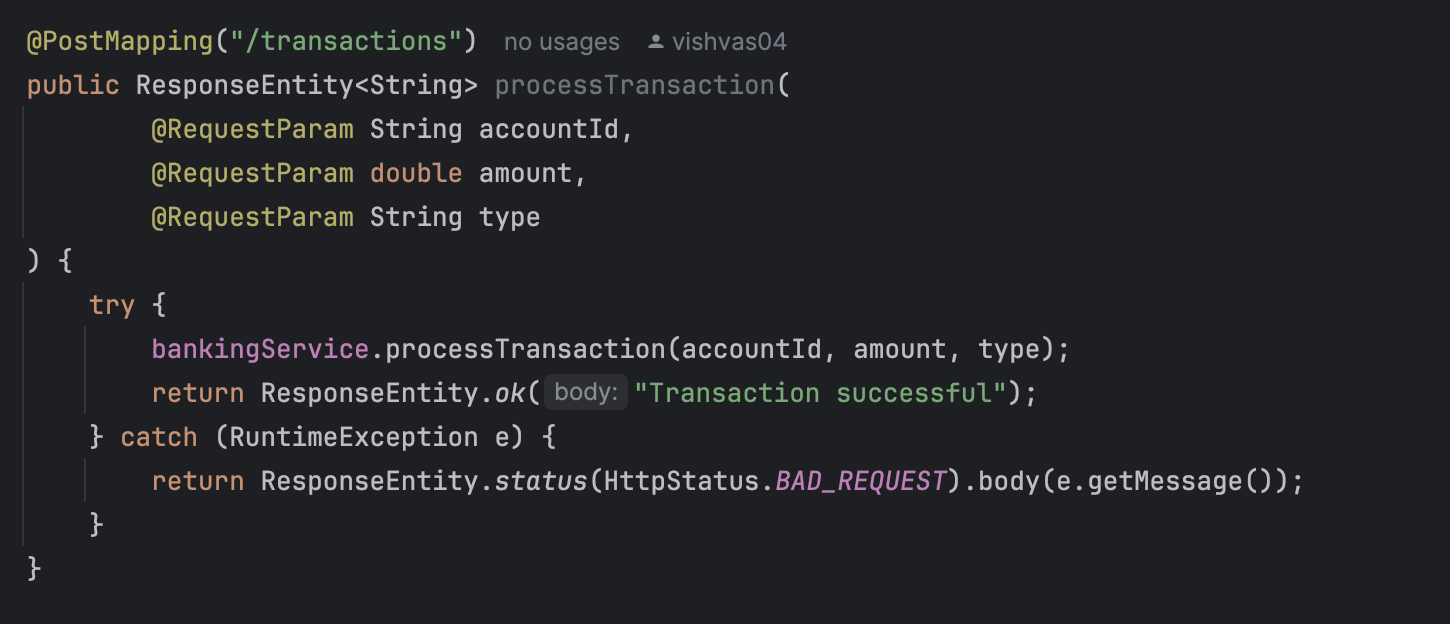
**POST /accounts —> For Account Creation**

****

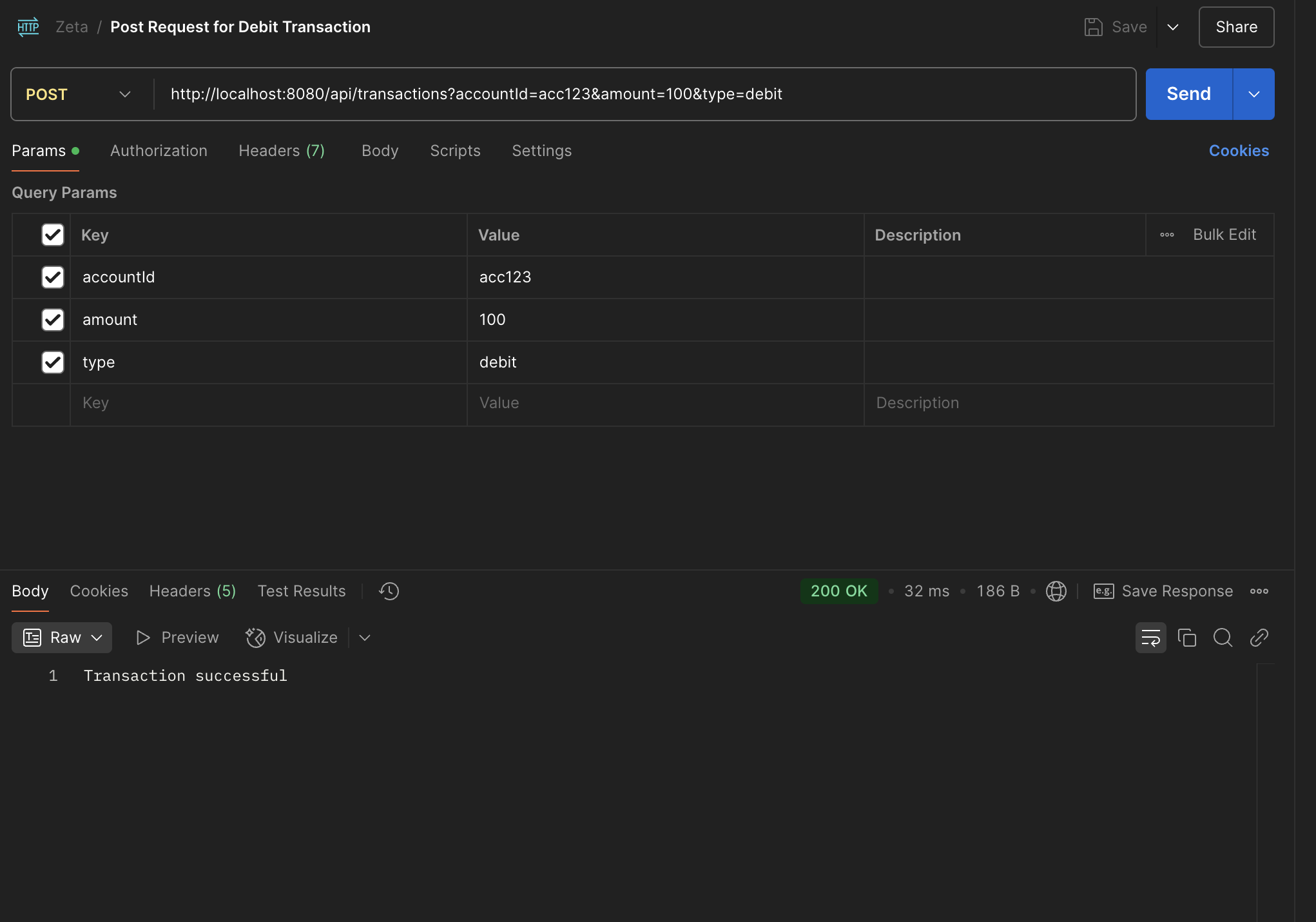
**Returns a response after successful creation of account:**

****

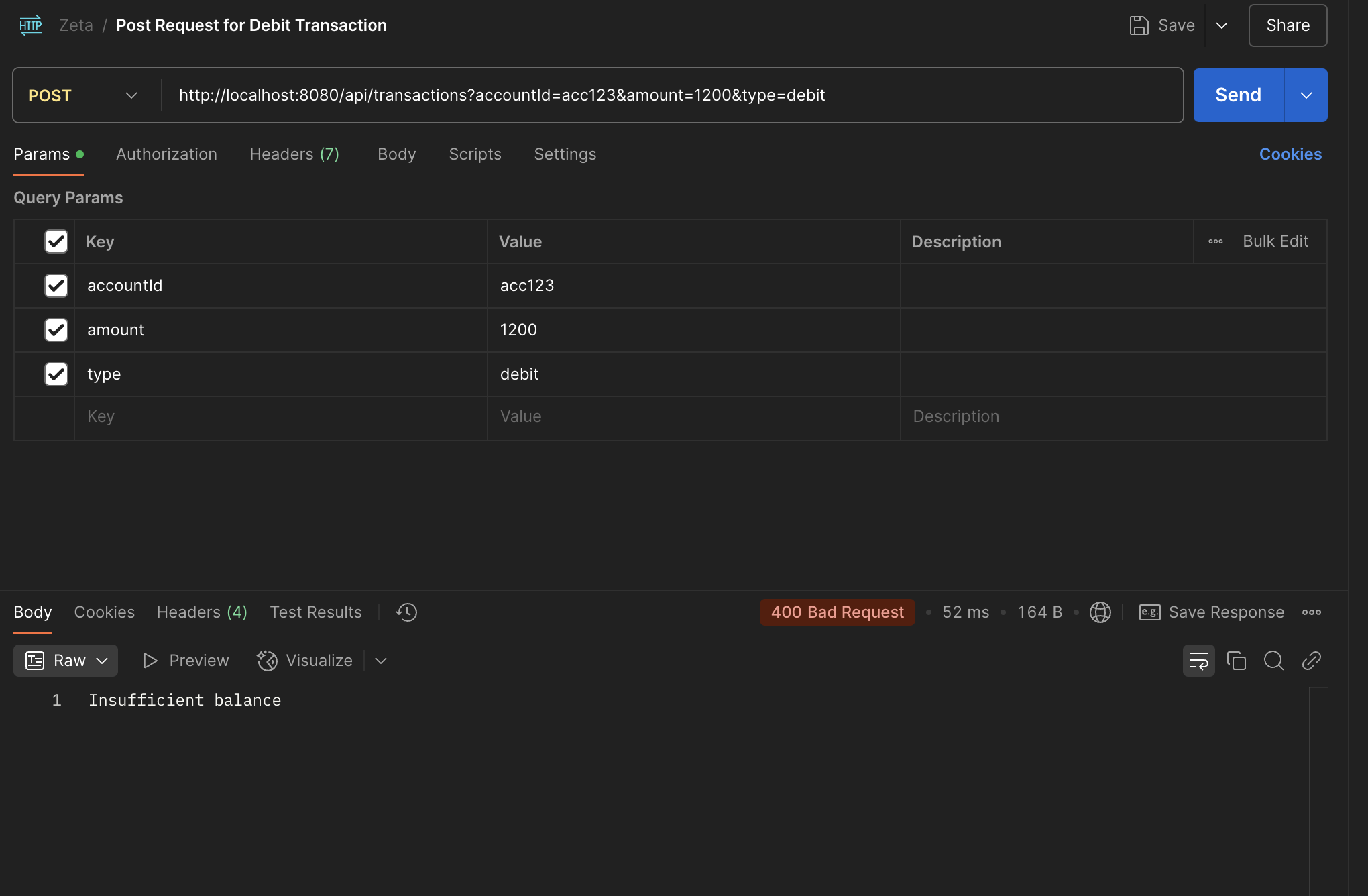
**POST /transactions —> For Debit transactions**

****

If the requested amount exists the transaction will be successful and return the message with 200 status code.

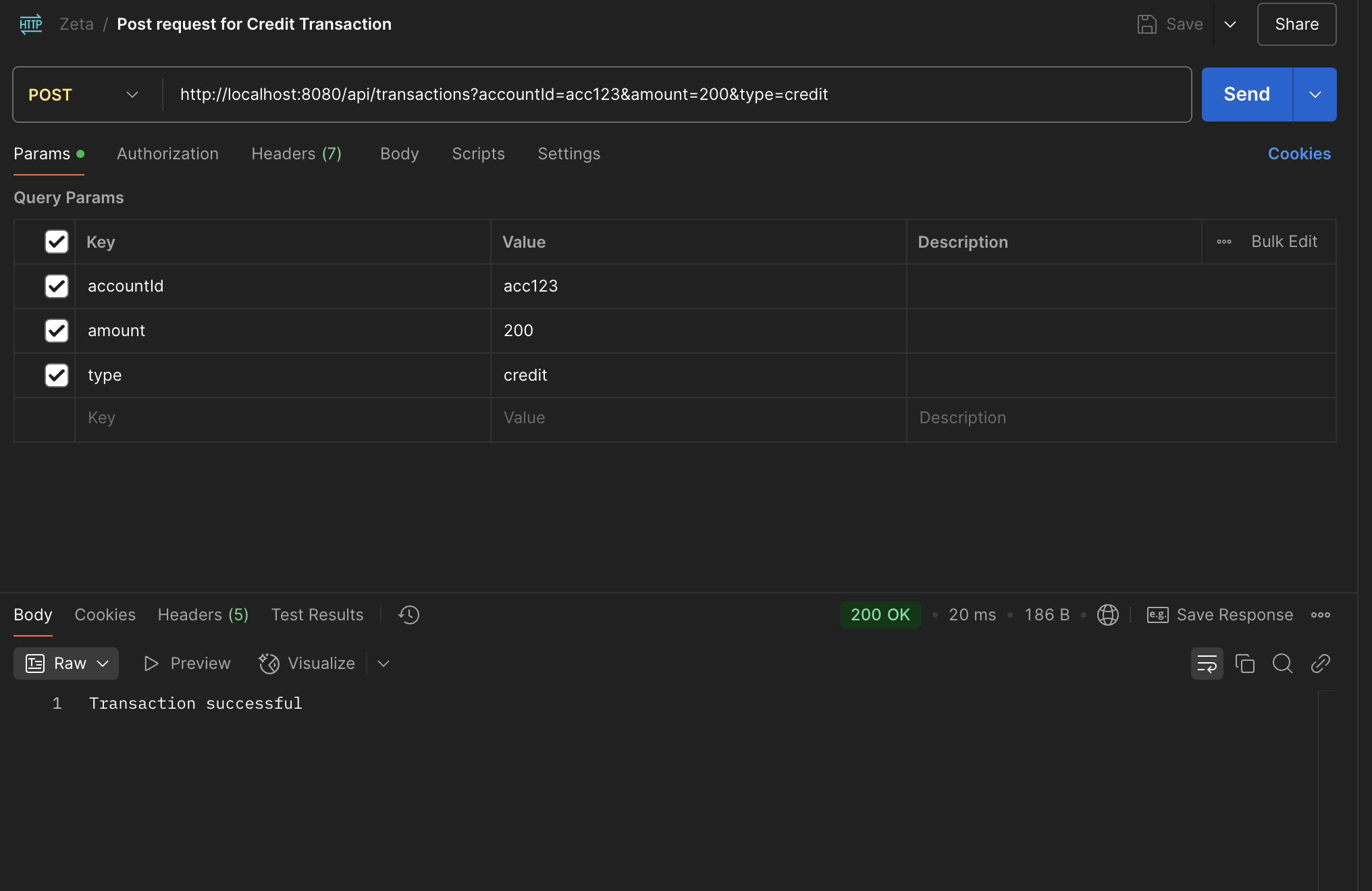
****

Since the amount in the account is 1000, if we are trying to withdraw amount greater than 1000 it is giving an error of insufficient balance.

****

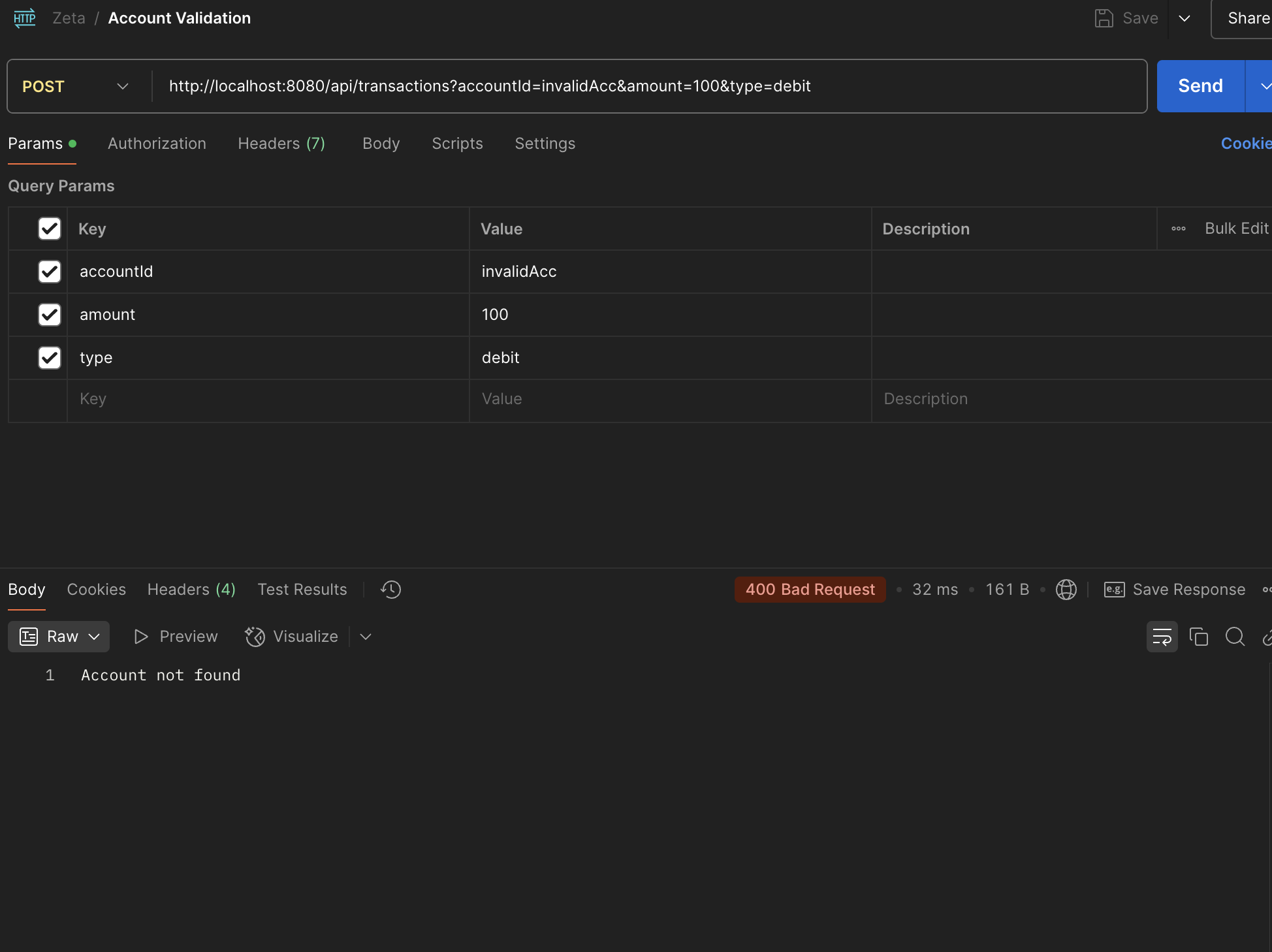
**POST /transactions → For Credit**

You can even add money to your account and return 200 code with successful transaction message.

****

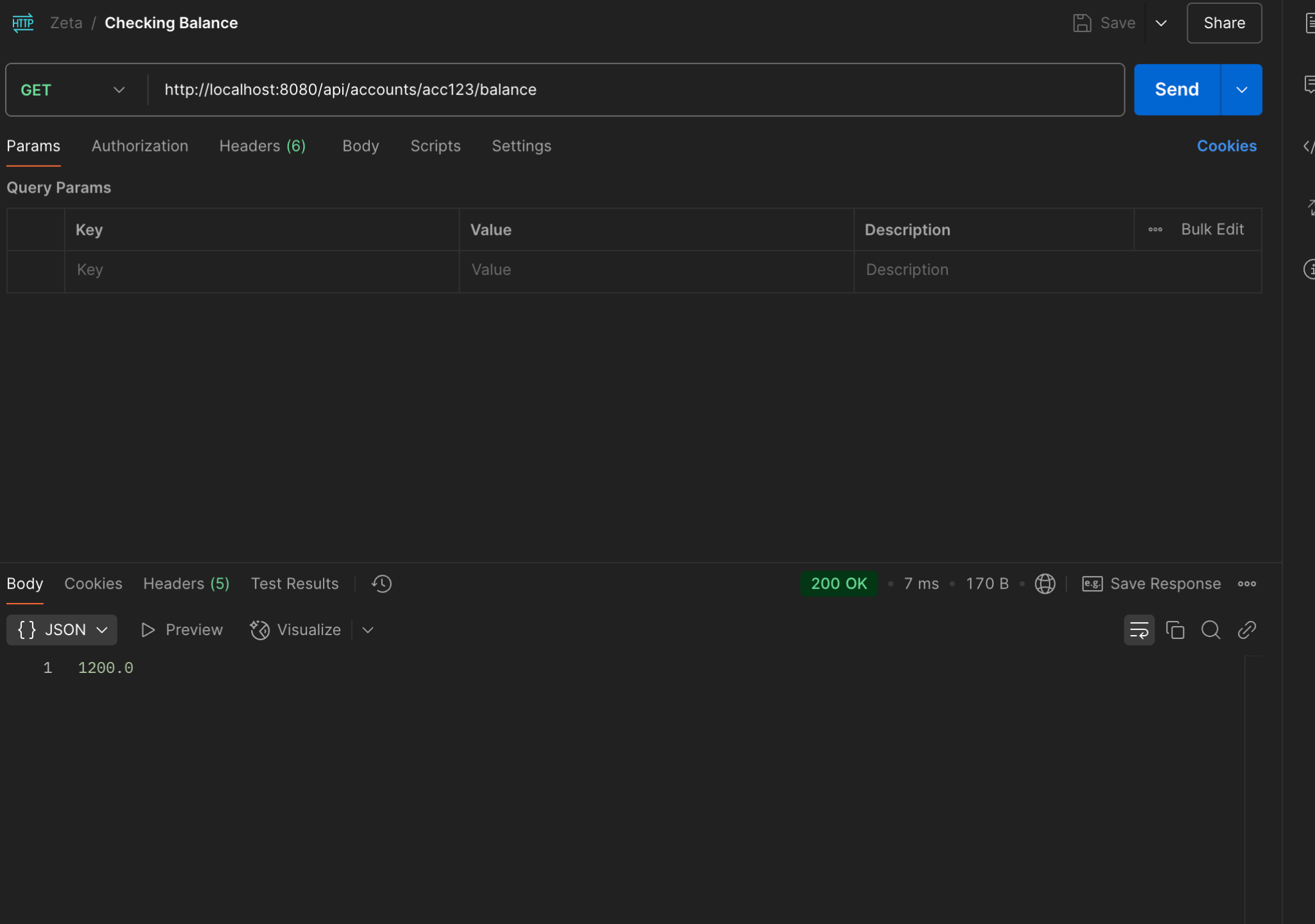
**POST → Account Validation**

Return Account does not exist when an account is not created yet.

****

**GET /accounts/{accountID}/balance**

It returns the outstanding balance present in the account:

****

**Error Responses:**

400 Bad Request: Insufficient balance.

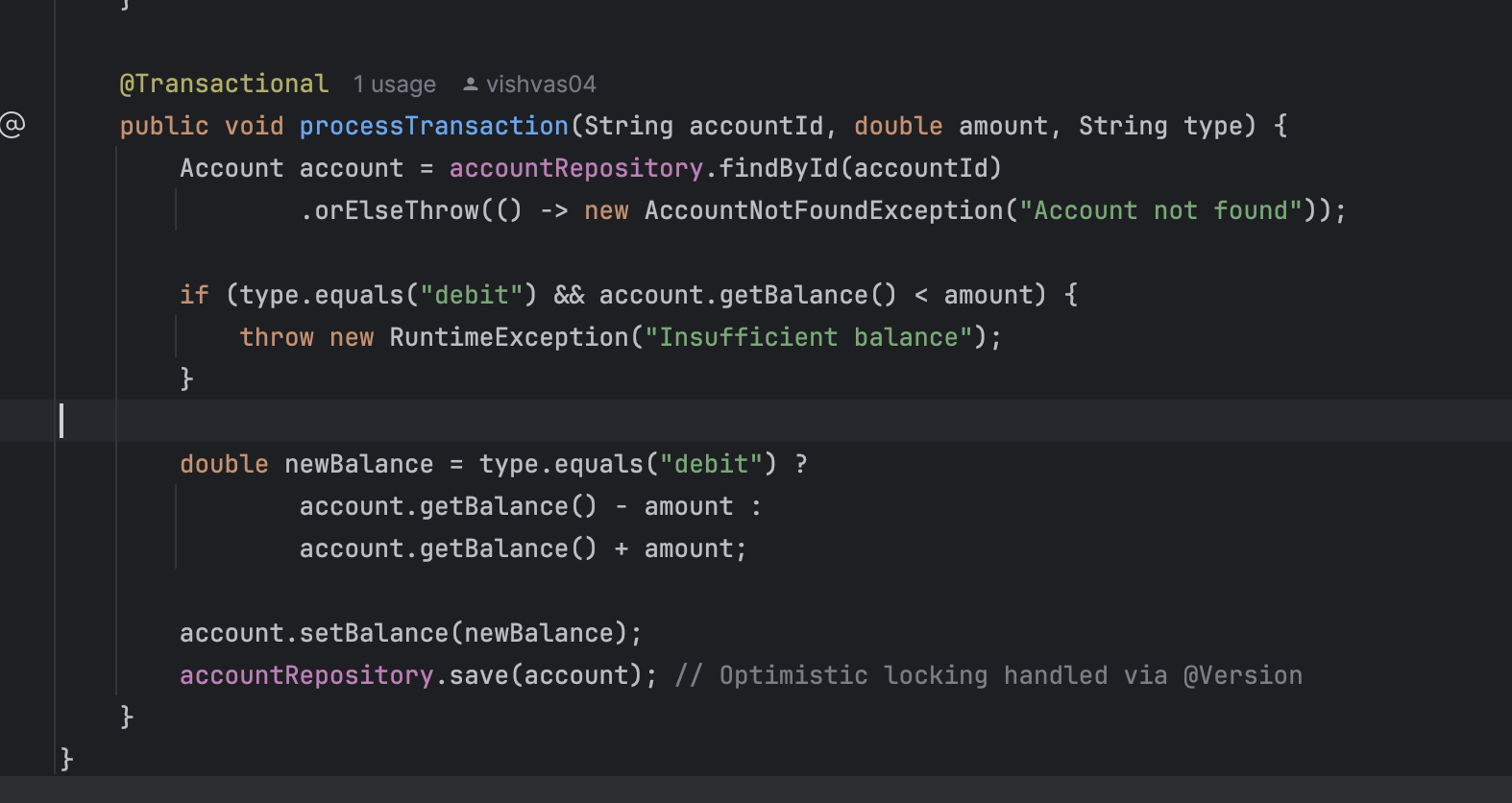
404 Not Found: Account not found.

409 Conflict: Concurrent modification detected.

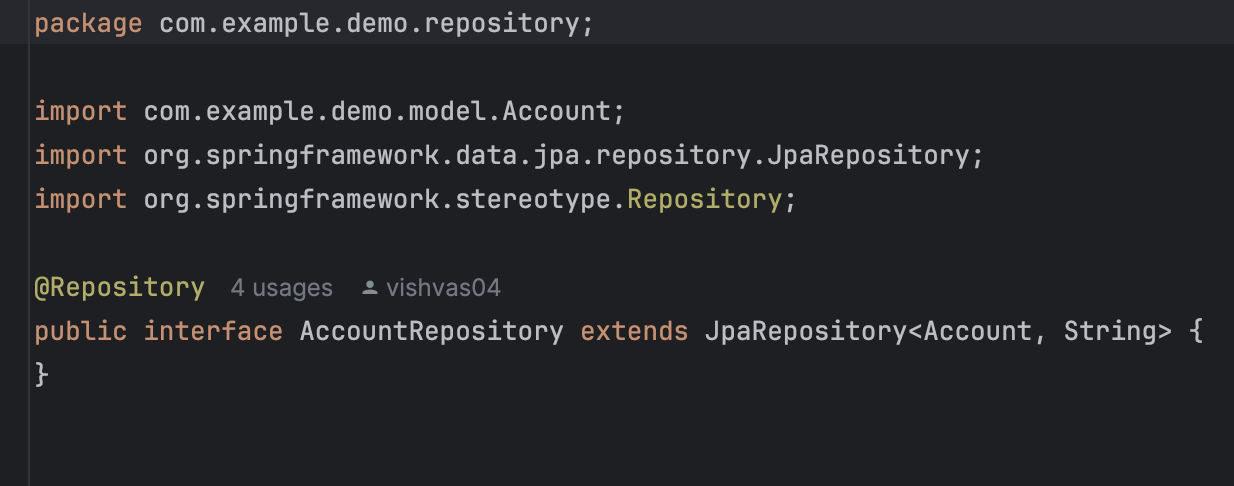
**Consistency & Concurrency Handling:**

To Ensure Atomicity and Consistency I am using Spring @Transactional where spring handles balance updates are made atomically.

For concurrency I am using the Optimistic locking where in the schema of Account I added a version variable where it maintains current condition in the database and transaction only happens only when the locking is removed.



I have added an Account Repository layer which extends the JPA Repository where it accounts methods like findByID() .save which is used for Database interactions and it keeps clean code.



And Finally to increase the performance optimisations I didn't go deep into this and I am exploring how we can make bulk insertions and also and also integrate message queuing for asynchronous operations using Apache Kafka.

I used H2 console for testing purpose as it provides in memory database for fast retrieval and I want to replace it with PostgreSQL as a Production Database

