# **Technical Document: Distributed Locking Microservice Solution**

## **Table of Contents**

1. **Executive Summary**
2. **Problem Statement**
3. **Solution Overview**
4. **System Architecture**
5. **Technical Implementation**
6. **API Documentation**
7. **Deployment Strategy**
8. **Monitoring and Maintenance**
9. **Security Considerations**
10. **Performance Considerations**
11. **Appendix**

## **1. Executive Summary**

This document outlines a centralized distributed locking microservice solution designed to prevent race conditions and deadlocks in payment processing systems across multiple Spring Boot pods. The solution utilizes pure Java collection features for application-level locking without database dependencies.

## **2. Problem Statement**

### **Current Challenge**

The payment processing system receives a high volume of concurrent requests from multiple Spring Boot pods. When multiple transactions target the same account simultaneously, race conditions occur, leading to:

* **Double spending** issues
* **Inconsistent account balances**
* **Deadlock situations**
* **Data integrity problems**

### **Use Case Example**

* **Pod 1**: Receives payment request for Account ID 100, Amount 1000
* **Pod 2**: Simultaneously receives payment request for Account ID 100, Amount 1000
* **Result**: Potential overdraft or inconsistent balance without proper locking

### **Requirements**

* Application-level locking using Java collections
* Centralized locking service
* Support for multiple client pods
* Automatic lock expiration
* Thread-safe implementation
* Minimal external dependencies

## **3. Solution Overview**

### **Architecture Approach**

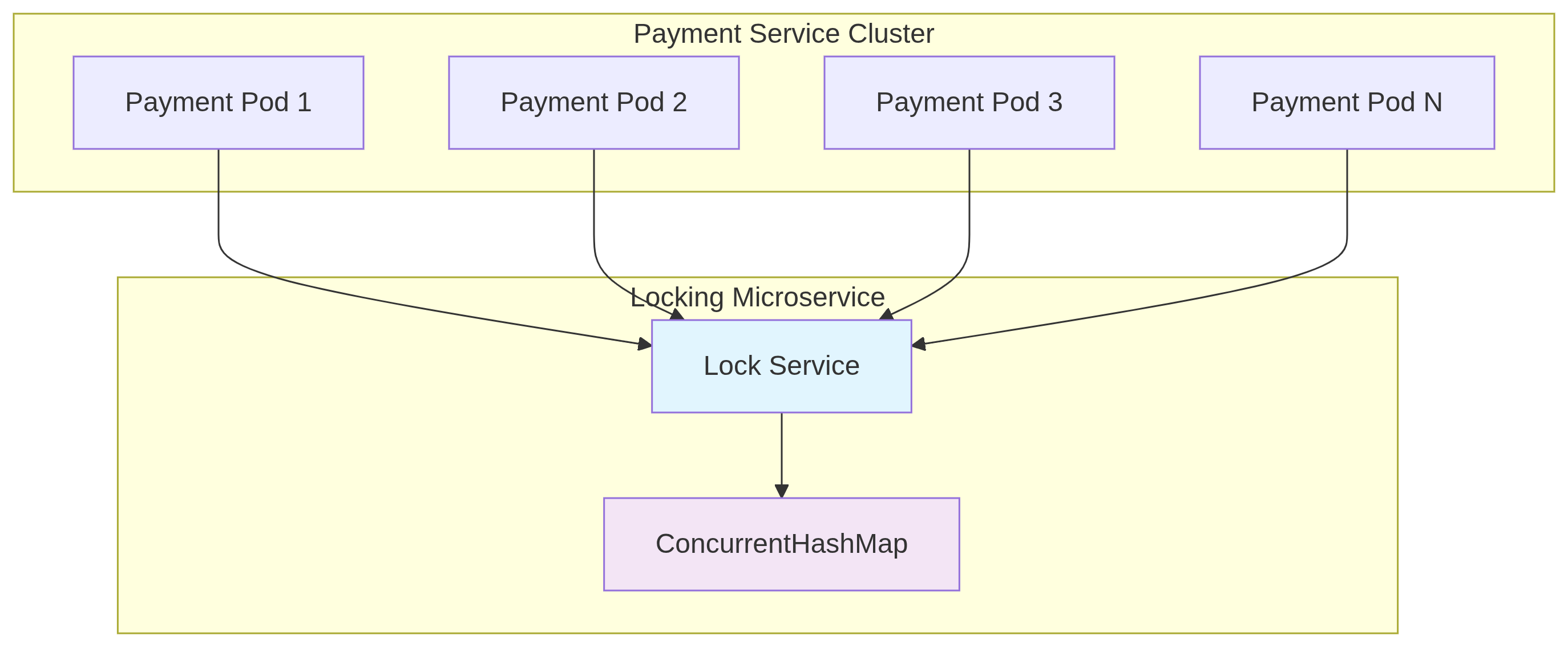
[Payment Pods] → [Central Locking Microservice] → [In-Memory Lock Storage]

### **Key Features**

* **Pure Java Implementation**: Uses ConcurrentHashMap for thread-safe operations
* **RESTful API**: Simple HTTP interface for lock management
* **Reentrant Locks**: Support for nested locking by same client
* **Automatic Cleanup**: Scheduled removal of expired locks
* **Timeout Mechanism**: Configurable lock expiration
* **Monitoring Endpoints**: Health checks and statistics

## **4. System Architecture**

### **4.1 High-Level Architecture Diagram**



\*Figure 1: High-Level System Architecture\*

### **4.2 Component Architecture**

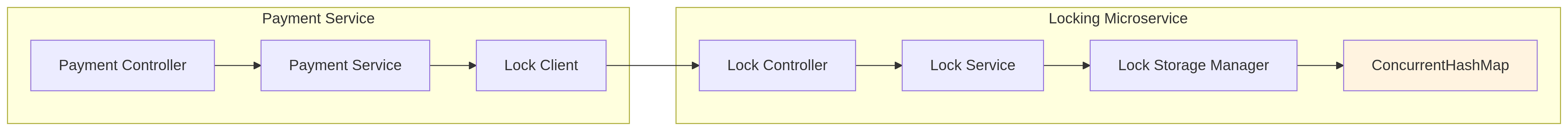


Figure 2: Component Architecture Diagram

### **4.3 Data Flow Diagram**

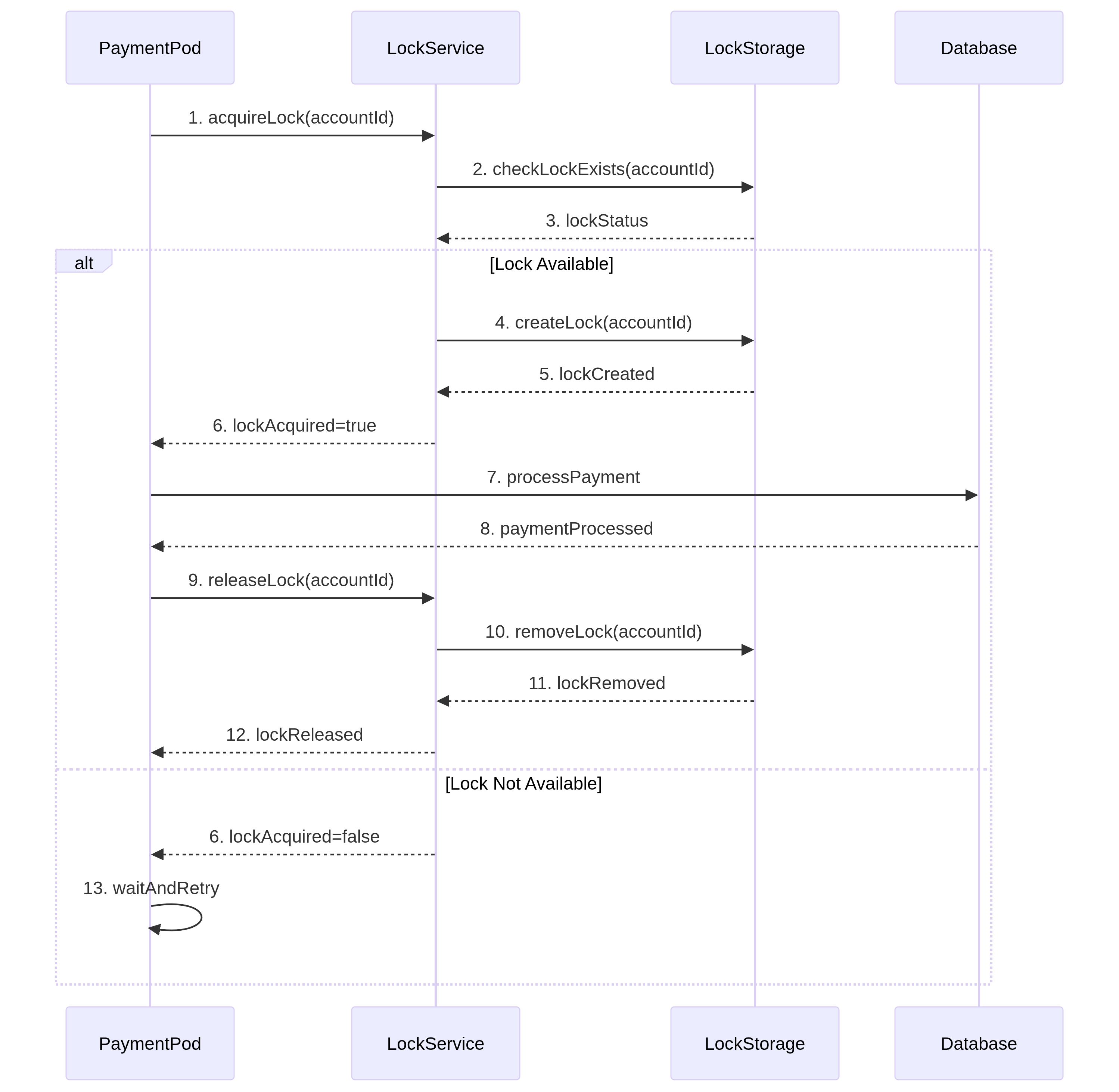


Figure 3: Data Flow Sequence Diagram

### **4.4 REST API Endpoints**

| **Method** | **Endpoint** | **Description** | **Parameters** |
| --- | --- | --- | --- |
| POST | **/api/locks/acquire** | Acquire a lock | **LockKey:-- account no & branch no**,  **value :-- Transaction** |
| POST | **/api/locks/release** | Release a lock | **LockKey:-- account no & branch no,**  **value :-- Transaction** |