Software Engineer - Technical Task

Context:

The current codebase for managing user sessions in a NoSQL database contains some logic that could be refactored for clarity, maintainability, and basic performance improvements. Your task is to refactor a provided code snippet, improve error handling, and make small optimizations where possible.

Initial Code (Simplified):

You are provided with the following existing code snippet, which is part of a larger, more complex system:

```
async function processSession(sessionId: string) {
  const session = await db.getSession(sessionId);
  if (!session) {
    throw new Error("Session not found");
  }

  const now = Date.now();
  if (session.lastActivityTimestamp + SESSION_TIMEOUT < now) {
    session.isActive = false;
    await db.deleteSession(sessionId);
    await db.addLog({ sessionId, logTimestamp: now, message: "Session expired" });
  } else {
    session.lastActivityTimestamp = now;
    await db.updateSession(session);
    await db.addLog({ sessionId, logTimestamp: now, message: "Session updated" });
  }
}
```

Data Structure:

```
Session table:
sessionId (Partition Key)
userId
lastActivityTimestamp
isActive (boolean)
sessionLogs (list of logs)

Log table:
logId (Partition Key)
sessionId
logTimestamp
message
```

Objective:

In this task, you will refactor the code to improve its readability, maintainability, and error handling while keeping optimizations simple and realistic within the time constraints.

Task Requirements:

1. Refactoring:

- **Modularization**: Break the provided `processSession` function into smaller, clearly defined functions. Each function should have a single responsibility, following basic **SOLID** principles.
- **Remove Hardcoded Values**: Ensure values like `SESSION_TIMEOUT` are configurable (e.g., as a function parameter or an environment variable).

2. Error Handling:

- Implement **basic error handling** to manage cases such as:
  - Session not found.
  - Expired session.
  - Any database-related issues (e.g., failure to delete a session or add a log).

3. Simple Optimization:

- **Minimize Redundant Database Calls**: Refactor the code to reduce the number of NoSQL database interactions where possible. You may assume database latency is a concern, so look for small optimizations like eliminating unnecessary writes.

Deliverables:

1. **Refactored Code**:
   - Submit your refactored TypeScript code as a single file in a private GitHub repository.
   - The code should be modular and easy to understand.
2. **Documentation**:
   - Provide a short **README** or markdown file (max. 300 words) explaining:
     - Key architectural decisions (e.g., why you modularized functions in a certain way).
     - Any optimizations you made to reduce database calls.
     - Basic assumptions you made (e.g., regarding how sessions should expire).