Spreadsheet application

1. Introduction

1.1 Purpose

The purpose of this document is to define the functional and non-functional requirements for a collaborative spreadsheet application. The application ensures that only one user has write access at any given time, while others can only read the sheet. If another user attempts to write while the sheet is locked, they will receive an error message indicating that another user is currently editing. The spreadsheet must also update in real-time for all users once the editing user completes their modifications.

1.2 Scope

This project aims to provide a spreadsheet-like interface with controlled concurrent access using the following technologies:

- DuckDB as the database for storing and updating spreadsheet data.
- Redis Streams to handle real-time user interactions and manage access control.
- Flask (Python) for the front-end application and API services.
- Kubernetes to handle scalability in case of a high number of concurrent users.

The system will ensure:

- A single-user write access mechanism.
- Real-time updates to all connected users.
- Synchronization between the spreadsheet and DuckDB.
- Efficient handling of concurrent user requests using Redis Streams.

1.3 Definitions, Acronyms, and Abbreviations

- DuckDB – An in-memory database optimized for analytical processing.
- Redis Streams – A message queuing system used for handling concurrent access.
- Flask – A lightweight Python web framework for API development.
- Kubernetes – A container orchestration platform for scalability and deployment.

1.4 References

- Redis Streams Documentation: https://redis.io/docs/streams/
- DuckDB Official Site: https://duckdb.org/
- Flask Documentation: https://flask.palletsprojects.com/

1.5 Overview

This document details the system features, functional and non-functional requirements, use cases, and constraints for the collaborative spreadsheet application.

2. Overall description

2.1 Product Perspective

This system is designed as a web-based collaborative spreadsheet application with controlled access, real-time updates, and high scalability. It will function as a standalone system that interfaces with DuckDB, Redis Streams, and Flask.

2.2 User Classes and Characteristics

- Standard Users: Users who can read and request write access to the spreadsheet.
- Admin Users: Users who can configure the application settings and manage user access policies.

2.3 Operating Environment

- Web-based interface accessible via modern browsers.
- Backend hosted on cloud or Kubernetes-managed infrastructure.
- Database operations performed using DuckDB.
- Real-time messaging and event handling via Redis Streams.

2.4 Design and Implementation Constraints

- The system must prevent concurrent write access to the spreadsheet.
- Data synchronization with DuckDB must be efficient and consistent.
- The system should support high concurrency via Kubernetes auto-scaling.
- Network failures should not lead to data loss or inconsistency.

3. Specific requirements

3.1 Functional requirements

3.1.1 Single User Write Access

- Only one user can have write access at a given time.
- If another user attempts to write, they receive an error message: "Oops! Someone else is updating the spreadsheet."

3.1.2 Read-Only Mode for Other Users

- Users without write access can view real-time updates but cannot modify data.
- Once the editing user completes their changes, the sheet updates for all users.

3.1.3 Real-Time Synchronization

- Changes made by the writer should be reflected instantly for all read-only users.
- Redis Streams will handle real-time communication and broadcasting updates.
- DuckDB will be updated immediately after every write operation.

3.1.4 Access Control & Lock Management

- When a user starts editing, they acquire a write lock.
- The lock is released once the user completes editing or times out due to inactivity.
- If a user disconnects unexpectedly, the system should release the lock within a defined timeout.

3.2 Non-Functional requirements

3.2.1 Performance

- The system should support at least 100 concurrent users with minimal latency

3.2.2 Scalability

- Kubernetes should allow automatic scaling of backend services based on demand

3.2.3 Reliability

- System should recover from network failures without losing data

3.2.4 Security

- User authentication and authorization should be enforced to control access

4. Use case scenario

4.1 Use case: editing a Spreadsheet

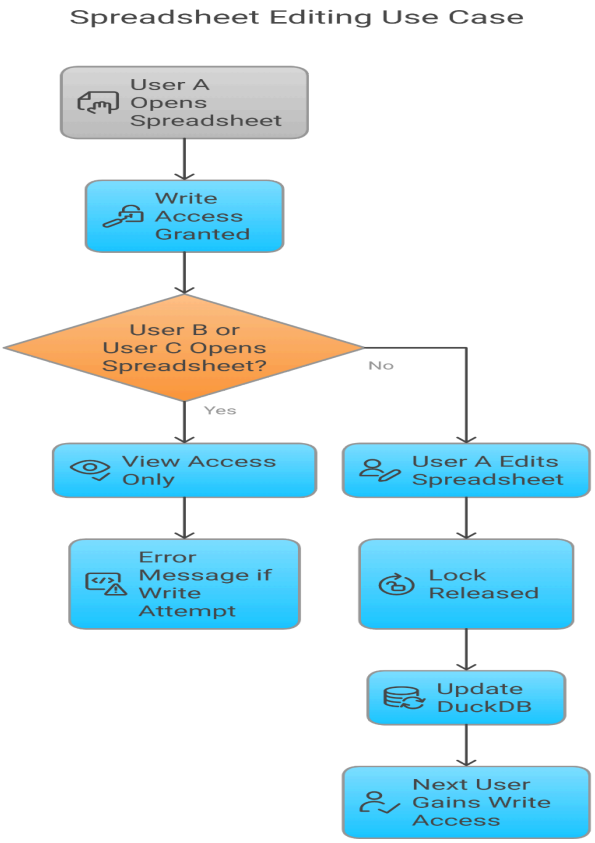
Actors: Standard User

Preconditions:

- The user is Authenticated
- The spreadsheet is accessible

Flow of Events:

- User A opens the spreadsheet and gains write access.
- User B and User C open the spreadsheet but can only view the data.
- If User B tries to write, they receive an error message: *"Oops! Someone else is updating the spreadsheet."*
- Once User A completes their edits, the system releases the lock and updates the sheet for all users.
- DuckDB is updated with the latest changes.
- The next user requesting write access can now edit the spreadsheet.



5. Expected challenges & considerations

- Ensuring write-lock consistency: The system should ensure that only one user gets the write lock at any given moment, preventing race conditions.
- Handling network failures: If the writing user disconnects unexpectedly, the lock should be released appropriately.
- Updating DuckDB efficiently: Ensuring that all changes made by the write-access user are committed correctly to DuckDB without data loss or corruption.
- Scaling with Kubernetes: Efficient handling of user requests when there are a large number of concurrent users.
- Real-time synchronization: Ensuring that all read-only users receive instant updates without significant delays.

6. Conclusion

This project aims to create a robust and scalable spreadsheet application that guarantees controlled write access while maintaining real-time synchronization across multiple users. By leveraging DuckDB, Redis Streams, Flask, and Kubernetes, we can achieve an efficient, real-time collaborative experience with proper access control while ensuring that the database remains updated in sync with the spreadsheet.