# - Collaborative Notepad -

Cornea Alexandra-Nicoleta

March 28, 2024

## 1 Introduction

The proposed project aims to develop a client/server application for simultaneous text file editing. The main purpose is to allow two clients to edit a single file at the same time and manage multiple concurrent editing sessions on different documents. An essential aspect is implementing a communication protocol to ensure the consistency of the edited document in case of simultaneous changes to a common part.

## 2 Applied Technologies

### 2.1 Communication Protocol: TCP

For implementing client/server communication, I chose to use the Transmission Control Protocol (TCP) in the C++ programming language. The choice of this communication protocol is due to its reliability and ensuring the order of data transmission, critical aspects for the consistency of simultaneously edited documents.

### 2.2 Framework + SQLite + OT

For client development, I used C++ together with the Qt framework to implement the graphical interface and event management. Additionally, I integrated the SQLite database for storing documents. To synchronize changes made by multiple users to a shared document in a way that maintains consistency, I used Operational Transformation (OT) technique. The `OTDocument` class is used for the basic implementation of the collaborative editing system.

#### Transform Function

The `transform` function takes a remote operation and transforms it based on local operations. It uses rules based on operation types ('I' for insert, 'D' for delete) and adjusts the position accordingly.

```
QString OTDocument::transform(const QString& remoteOperation) {
    // ...
    for (const Operation& localOperation : operations) {
        // Apply transformation rules based on operation types
        if (opType == 'I' && localOperation.type == 'I') {
            if (remotePosition > localOperation.position ||
                (remotePosition == localOperation.position && opType
                   == 'I')) {
                //remotePosition += localOperation.text.length();
            }
        } else if (opType == 'D' && localOperation.type == 'I') {
            if (remotePosition >= localOperation.position) {
                remotePosition -= localOperation.text.length();
            }
        } else if (opType == 'I' && localOperation.type == 'D') {
            if (remotePosition > localOperation.position) {
                remotePosition += localOperation.text.length();
```

```
            }
        }
    }
}
```

## C++ Function `sendDocumentList`

The `sendDocumentList` function seems to be a C++ function that communicates with an SQLite database to retrieve a list of documents and send it to a client through a socket connection. Here's an explanation of the code:

```
void sendDocumentList(int clientSocket) {
    std::string message = "";

    loadDocumentsFromDatabase();

    sqlite3_stmt* statement;
    std::string query = "SELECT id, document_name FROM Docs";
    if (sqlite3_prepare_v2(db, query.c_str(), -1, &statement, nullptr)
        == SQLITE_OK) {
        while (sqlite3_step(statement) == SQLITE_ROW) {
            const unsigned char* documentName = sqlite3_column_text(
                statement, 1);
            if (documentName) {
                message += reinterpret_cast<const char*>(documentName)
                    + std::string(",");
            }
        }
    } else {
        std::cerr << "Error: " << sqlite3_errmsg(db) << std::endl;
    }

    if (send(clientSocket, message.c_str(), message.size(), 0) == -1)
       {
    }
}
```

# 3 Application Structure

**Document Structure:**

- The document is represented by a string of characters, and the initial state of the document is provided when constructing the `OTDocument` object.

**Operations:**

- Document operations include inserting a character ('I') and deleting a character ('D').

**Operation Transformation Logic:**

- The `transform()` function is responsible for transforming a remote operation based on local operations. It decides the type of transformation (insertion or deletion) and adjusts the position based on local operations.

**Handling Concurrent Operations:**

- The `applyRemoteOperation()` function applies the remote operation to the document after it has been transformed. Transformation occurs to ensure consistency.

**Applying Local Operations:**

- The `applyOperation` function applies an operation to a given document, whether it is an insertion or deletion.

**Conflict Resolution:**

- The `transformInsert()` and `transformRemove()` functions handle potential conflicts between remote and local operations by adjusting positions and lengths.

**Communication:**

- Communication between users is based on transmitting and applying remote operations. This process is exemplified in the `applyRemoteOperation()` function.

**Testing:**

- The code needs to be tested for various scenarios, including concurrent edits, to ensure the correctness and convergence of the transformation and operation application algorithms.

The project is structured into a server and a client that communicate via sockets. After establishing the connection, the server creates threads for each client. Figure 1.

# 4 Implementation Aspects

## 4.1 Application-level Protocol

The application-level protocol implemented for simultaneous text file editing between clients and server aims to manage concurrent operations on documents. The protocol ensures efficient communication between clients and servers to maintain the consistency of documents edited simultaneously. Below are the details of the protocol:

1. Server Connection Request (Client): The client sends a connection request to the server, also requesting the list of available documents. Response (Server): The server queries the database and sends an acceptance response, allocating a session to the client and the list of documents.

2. Initial Synchronization Request (Client): The client requests the current content of the document and session information (cursor position, selections, etc.). Response (Server): The server sends the document content and information about other active sessions.

3. Asynchronous Modifications Notification (Client): The client sends notifications about local changes made to the document (insertion, deletion, cursor movement, etc.). Notification (Server): The server broadcasts the changes to all connected clients for that document.

4. Temporary Locking Notification (Server): The server can send a temporary locking notification to clients to avoid conflicts with simultaneous modifications. For example, when a client wants to save a local document.

5. Disconnection Request (Client): Upon disconnection, the client sends a session closure request. Response (Server): The server releases the resources associated with the session and sends a disconnection confirmation.

6. Session Management Notification (Server): The server notifies clients about changes in concurrent sessions (e.g., another user has connected or disconnected).

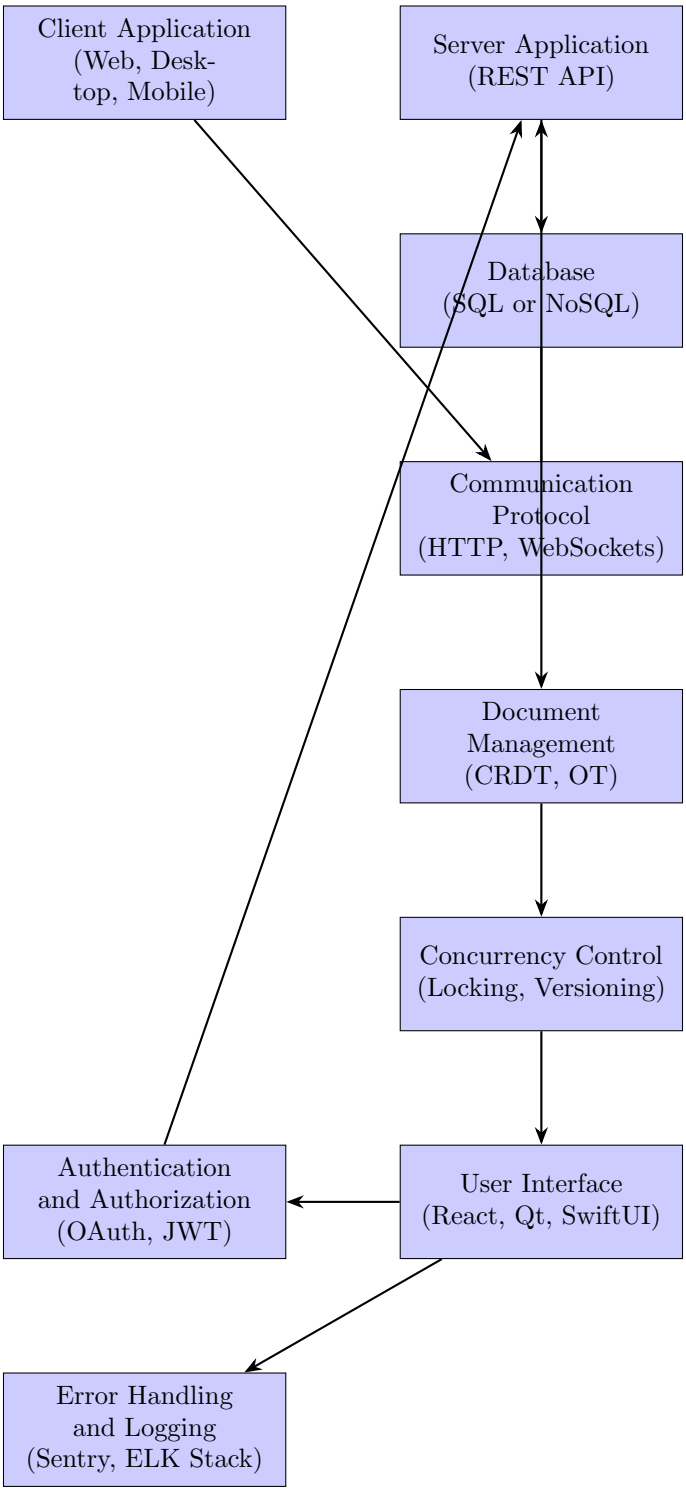


Figure 1: Caption here

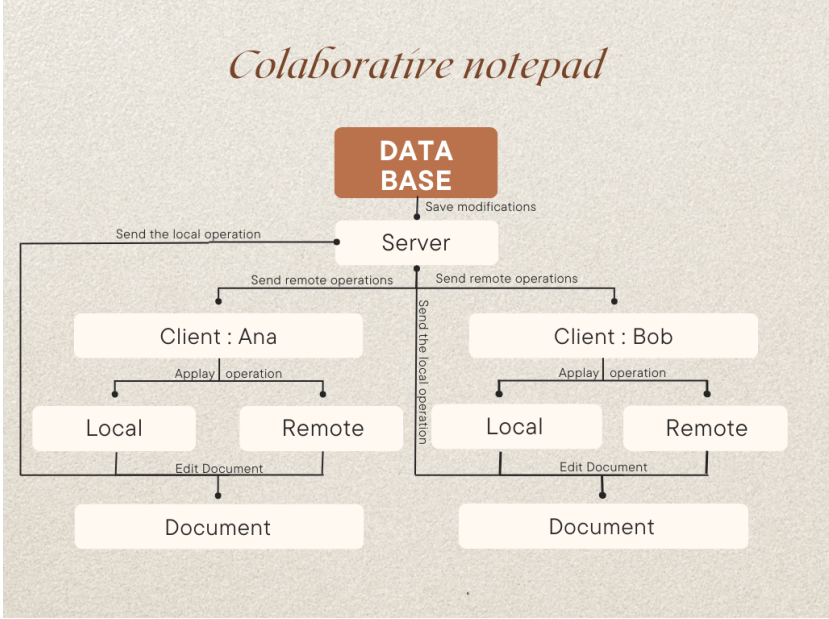


Figure 2: Application-level Structure

7. Conflict Resolution Notification (Server/Client): In case of detecting a conflict (simultaneous modifications at the same location), the server or client can send conflict notifications to be resolved.

   The protocol is designed to efficiently handle concurrent operations, maintain document consistency, and provide a smooth editing experience for users.

### 4.2 Usage Scenarios

1. Server Connection: Users connect to the server and receive a list of available documents, from which they select a document. There's also an option for creating a new document, for which the server creates a new document and saves it in the database.

2. Simultaneous Editing: After selection, the user edits simultaneously with another user who selected the same document. The changes are reflected in real-time for both users.

## 5 Conclusions

The project provides an efficient solution for simultaneous text file editing, implemented in C++, and using the Qt framework for the client interface. For future improvements, we can explore implementing functionalities for online sharing and collaboration to allow multiple users to edit the same document simultaneously, as well as optimizing server performance to handle an increased number of sessions.

## 6 References

## References

[1] Leslie Lamport, *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, 1994.

[2] Bjarne Stroustrup, *Programming: Principles and Practice Using C++*. Addison-Wesley, 2013.

[3] The Qt Company, *Qt Framework*. https://www.qt.io/.

[4] Mike Owens, *The Definitive Guide to SQLite*. Apress, 2006.

[5] Canva online graphic design platform, *The Canva editor*. https://www.canva.com/. June 28, 2012