Report

Date: 23.01.2026

1 Code Architecture and Refactoring

In the current stage of work, a thorough refactoring of the application structure was carried out, eliminating technical debt and code duplication. The GUI library was also changed from `tkinter` to `PyQt6`, which allowed for a more modern interface.

Updated module structure:

- **editor.py (Base GUI Layer):**
  - Defined `BaseTextEditor` class inheriting from `QWidget`.
  - **Interface logic centralization:** This class is responsible for all visual elements (toolbars, text editor, theme support) and file operations (I/O).
  - **Duplication elimination:** Methods responsible for appearance are no longer duplicated in derived classes.
- **concurrency.py (Distributed Logic Layer):**
  - The `ConcurrentTextEditor` class now inherits directly from `BaseTextEditor`.
  - **Separation of concerns:** This class extends the base editor solely with network mechanisms (UDP Socket), listener thread management, and state synchronization logic (CRDT).
  - **Implementation of "Snapshot" mechanism:** Added functionality to send the full editor state to newly joining users.
- **crdt.py (New algorithmic module):**
  - Implementation of the RGA (Replicated Growable Array) data structure.
  - The module is responsible for the mathematical model of resolving editing conflicts (inserting and deleting characters based on unique node identifiers, not absolute indices).

2 Communication and Algorithmics

The data synchronization model was changed from simple text overwriting to an incremental model.

Conflict Resolution (CRDT):

- **Problem:** In the previous version, simultaneous editing in the same place led to "letter mashups" or work overwriting.
- **Solution:** Applied a Conflict-free Replicated Data Type algorithm. Each character in the editor has a unique identifier (Lamport Timestamp + Client ID).

- **Effect:** Users can write simultaneously on the same line without the risk of data loss; everyone has a virtual, independent cursor.

Network Protocol:

- **Data format:** Instead of sending raw text (String), serialized operations are sent in JSON format (e.g., `CRDT_INSERT`, `CRDT_DELETE`).
- **Optimization:** Larger packets (e.g., during initial synchronization) are compressed using the Gzip algorithm and encoded in Base64, and split into smaller fragments (Chunking) if they exceed the MTU size.

3 Technical Solutions and New Functionalities

A number of utility and technical improvements have been implemented:

- **Application icon change:**
  - Implementation of a dedicated application icon (change from default to custom).
- **Theme Support (Theming):**
  - Implemented a dynamic application style change system (6 themes, including Dark Grey, Warm Cream).
  - Used QSS (Qt Style Sheets) for interface color management.
- **Internationalization:**
  - Unified the presentation layer – all messages, menus, and dialog boxes have been translated into English.
- **Process Management (Windows):**
  - Added `AppUserModelID` support in Windows, which fixed the issue of missing dedicated icon on the taskbar.

4 Verification and Testing

System tests were conducted in a distributed environment.

Test Configuration:

- **Hardware:** 4 physical workstations.
- **Network:** Tests in 2 independent LAN networks and in a mixed configuration.

Results:

- Confirmed stability on operating systems: Windows, Linux, macOS.
- Verified correctness of the CRDT algorithm – no collisions when writing simultaneously by 4 users.
- Confirmed correctness of the mechanism for joining an ongoing session (downloading edit history).

5 Identified Issues

During testing, one functional bug was detected in the GUI layer:

- **Bug: Font attributes reset.**
  - **Description:** Changing the graphical theme causes the font type and size to revert to defaults, ignoring settings previously selected by the user.
  - **Cause:** Overwriting `QTextEdit` widget styles by the stylesheet without preserving the `QFont` object state.

6 Future Plans (Next Steps)

- Fixing the font reset bug when changing themes.