DATA STRUCTURES AND ALGORITHMS

[CSE331s] XML EDITOR

FINAL SUBMISSION

PREPARED BY

Farah Haitham Saddik Abd Elmaged	2200485
Mohammed Yasser Said Mohammed Braka	2200417
Karim Hosam Ahmed Ali	2201405
Omar Abdelgaber Elsayed	2101048
Maya ahmed farahat	2200144
Mohamed Ashraf Mohamed Mounir	2200597
Radwa Yasser Ahmed Abdelhamed	2200239

GITHUB LINK: github

Abstract

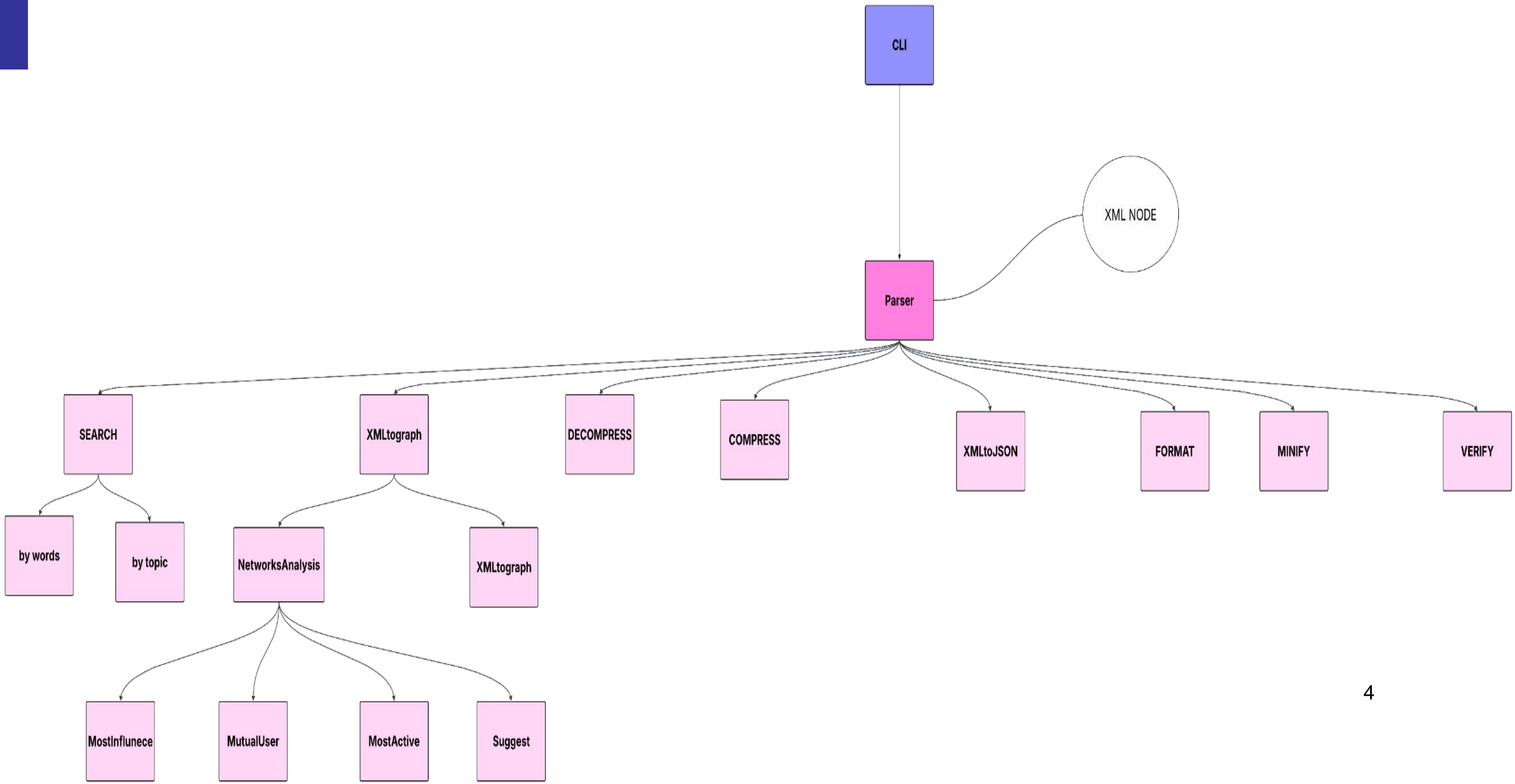
This project implements an XML editor utility suite in C++, providing parsing, formatting/minifying, JSON conversion, compression/decompression, error handling, and network-analysis features built on a custom `XmlNode` tree representation. The CLI exposes commands for transformation and analysis (format, mini, json, compress/decompress, verify, draw, suggest, most_influencer, most_active, mutual, search). This report documents architecture, data structures, implementation details, complexity analysis, tests and recommended future work.

Background

Implemented in C++ using a lightweight, custom parser. The project has 2 modes the first command-line oriented, with sample inputs in the `inputfiles/` directory and outputs in `outputfiles/`, and a second mode which is gui mode made with Qt creator.

System Architecture & Data Structures

project components and structure:



Primary data structure— `XmlNode`:

```
struct XmlNode {
    string name;
    string value;
    vector<XmlNode*> children;
    map<string,string> attrs; // attribute name -> value
    XmlNode() : name(""), value("") {}
};
```

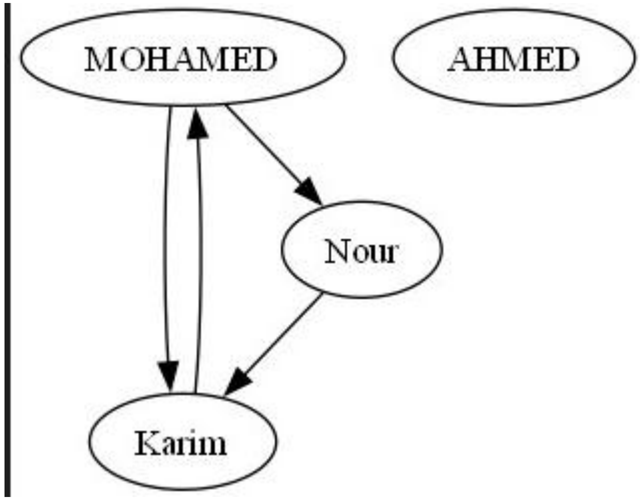
- Use:

  XML is inherently hierarchical; using a recursive tree traversal (DFS) for formatting ,conversion and searches

Graph/network structures

struct user and struct post:

```
class post {
public:
    string body;
    vector<string>topics;
};
class user {
public:
    int id;
    string name;
    vector<post>posts;
    vector<int>followers;
    user();
};
```



The graph is constructed as adjacency relations between user IDs in map<int,user> users in XML_to_graph.cpp

Implementation details and how to run:

how to compile:

Type : (g++ -std=c++17 -g (Get-ChildItem -Recurse -Filter *.cpp | ForEach-Object { $_.FullName }) -I. -o xml_editor.exe)

Then type ./xml_editor.exe (whatever command u chose)

those are the commands available:

```
//xml_editor json -i input_file.xml -o output_file_json.json

//xml_editor mini -i input_file.xml -o output_file_minified.xml

//xml_editor compress -i input_file.xml -o output_file_compressed.comp

//xml_editor decompress -i output_file_compressed.comp -o output_file_decompressed.xml

//xml_editor format -i inputPrettify.xml -o output_file_prettified.xml

//xml_editor verify -i inputErrorHandling.xml -f -o output_file_without_errors.xml
```

```
//xml_editor draw -i inputNetworkAnalysis.xml -o output_file_graphviz.jpg

//xml_editor suggest -i inputNetworkAnalysis.xml -id 1

//xml_editor most_influencer -i inputNetworkAnalysis.xml

//xml_editor mutual -i inputNetworkAnalysis.xml -ids 1,2

//xml_editor mutual -i inputNetworkAnalysis.xml -ids 1,2,3

//xml_editor most_active -i inputNetworkAnalysis.xml

//xml_editor search -w word -i inputNetworkAnalysis.xml

//xml_editor search -t topic -i inputNetworkAnalysis.xml

//xml_editor search -w lorem -i inputNetworkAnalysis.xml

//xml_editor search -t economy -i inputNetworkAnalysis.xmls
```

parser:

- - Algorithm (brief): tokenization is single-pass over the file, emitting tags and text tokens. `parse_node` implements a recursive-descent approach: on an opening tag it creates a node, then repeatedly parses child nodes or text until the matching closing tag is found; text tokens produce leaf nodes with `value` populated.
  - Robustness: minimal validation; the implementation expects reasonably well-formed tags. The `verify` command is used to detect/correct simple errors (see `errorHandling/ErrorHandling.cpp`).
  - Snippet (parsing loop pattern): `XmlNode* root = parse_node(tokens, idx);

Format / Prettify:

- File: [Format/FormatXml.cpp]
- Behavior: Depth-first traversal that prints opening tags with indentation, text content, and closing tags. Preserves textual content while reflowing tags for human readability.
- Algorithm: Depth-first traversal (DFS) of the `XmlNode` tree; at each node, print indentation proportional to depth, opening tag, then either the node's text or recursively its children, followed by the closing tag.

JSON conversion:

-File: json/convertxmltojson.cpp

- Policy: Elements become objects; repeated child elements can be represented as arrays; text content mapped to either a value field or node text depending on context.
- Algorithm: Walk the `XmlNode` tree recursively; for each node, collect child names and group repeated names into arrays. If a node has only text, map it to a string value; otherwise emit an object with child fields. Serialization is performed during traversal to avoid a separate in-memory representation where possible.

Compression / Decompression:

- File: compression/compress.cpp
  - Description: The project implements an LZW-style dictionary coder (see file for exact code). The compressor seeds a dictionary with the 256 single-byte sequences, scans the input bytes to emit the longest-match codes, and grows the dictionary dynamically. Compressed output is written as 16-bit codes (`unsigned short`). The decompressor rebuilds the dictionary while expanding codes and handles the standard LZW special case when a code equals the next dictionary index.

  - Practical notes: the implementation reserves dictionary capacity (4096 entries) and uses `unsigned short` codes (16-bit), so the dictionary is naturally bounded. This LZW-style approach is lossless and works well on repetitive XML, though production compressors (zlib/DEFLATE) may yield better ratios and streaming support.
  - Algorithm: `compress_helper` implements LZW: keep a current string `s`, append next character `c` to form `s+c`; if `s+c` exists in the dictionary, set `s = s+c`; otherwise output code for `s`, add `s+c` to dictionary, and set `s = c`. At end, output code for remaining `s`. Decompression mirrors growth of the dictionary, reconstructing strings for codes and appending to the output buffer; special-case when a code equals the current dictionary size is handled by repeating the previous entry's first character.

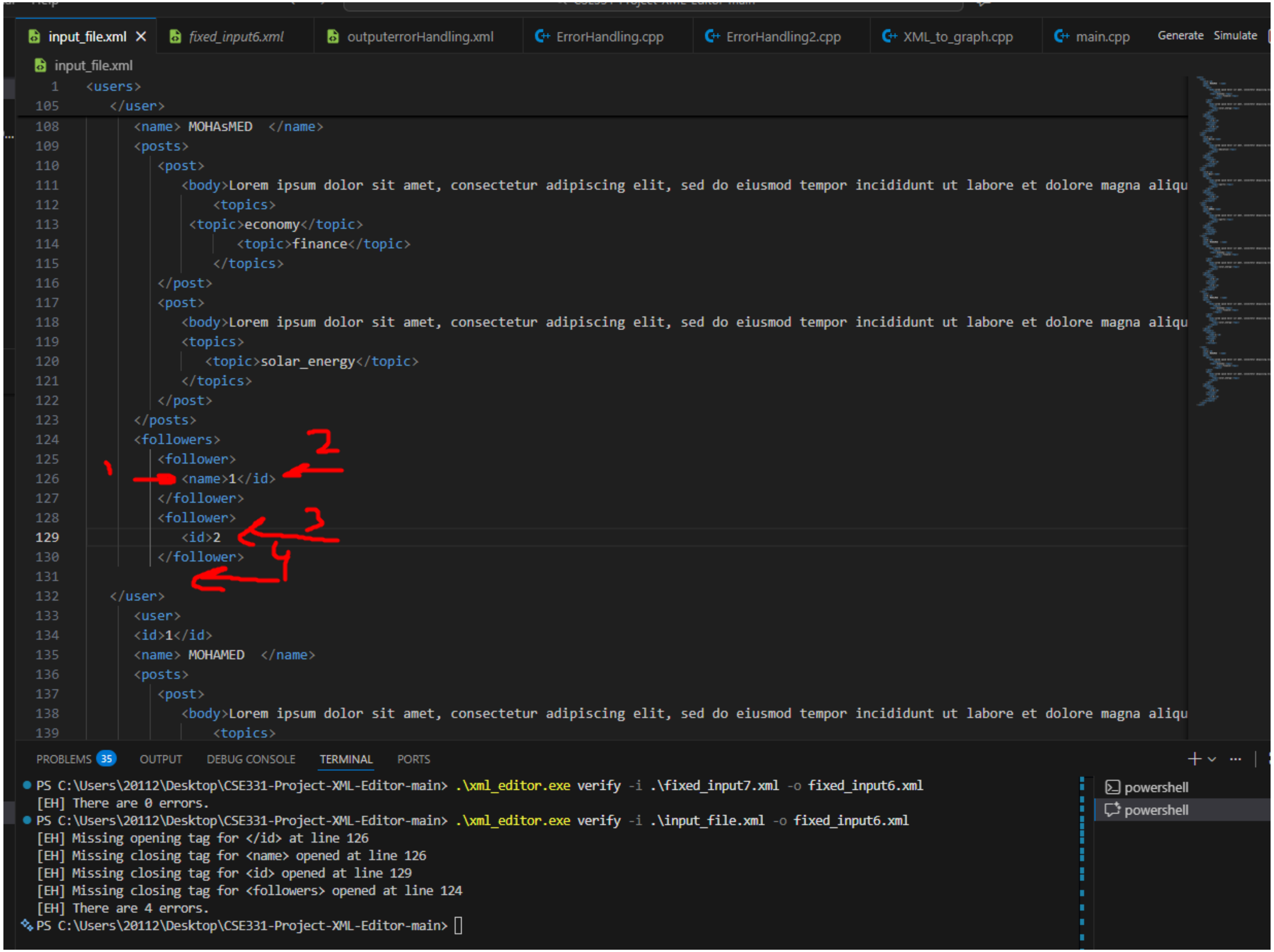
Compression ratio:

in the video wetried to compress large_network.xml which has 1379kb which became outlarge.comp with 77kb

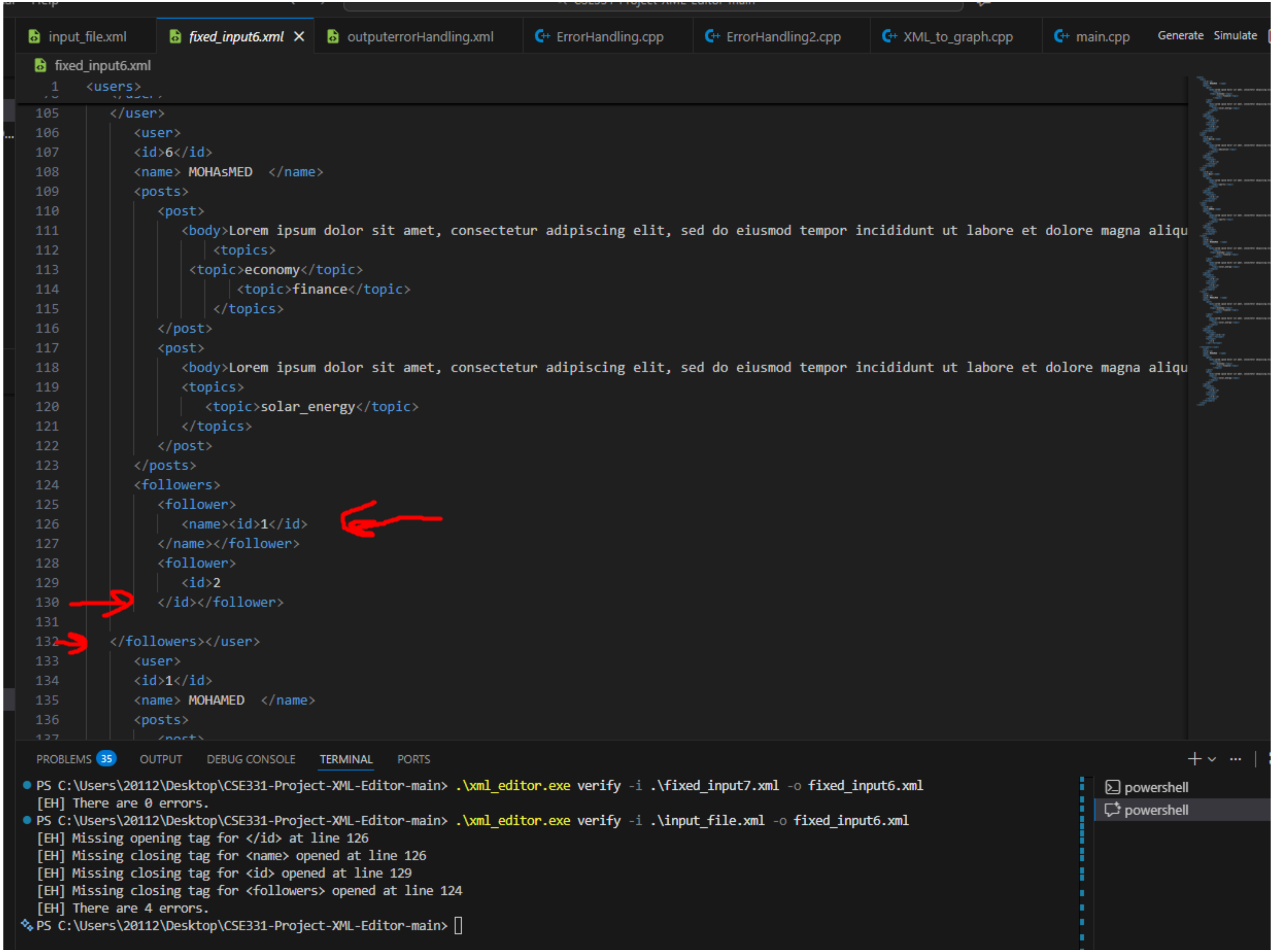
1379/77->1791% compression ratio

Error handeling:

- File: [errorHandling/ErrorHandling.cpp]
  - Function: `error_handling(input, output)` detects mismatched tags, missing closures, and produces a corrected `output_file_without_errors.xml` when possible.
  - Algorithm: read tokens and maintain a tag stack; on an opening tag push tag name, on a closing tag check top of stack — if it matches pop; if not, attempt local correction strategies (insert missing closing tags) and write a corrected stream. Simple heuristics are used (stack-based validation)



input_file.xml
fixed_input6.xml
outputerrorHandling.xml
ErrorHandling.cpp
ErrorHandling2.cpp
XML_to_graph.cpp
main.cpp
Generate
Simulate
input_file.xml
1 <users>
105 </user>
108 <name> MOHAsMED </name>
109 <posts>
110 <post>
111 <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
112 <topics>
113 <topic>economy</topic>
114 <topic>finance</topic>
115 </topics>
116 </post>
117 <post>
118 <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
119 <topics>
120 <topic>solar_energy</topic>
121 </topics>
122 </post>
123 </posts>
124 <followers>
125 <follower>
126 <name>1</id>
127 </follower>
128 <follower>
129 <id>2
130 </follower>
131
132 </user>
133 <user>
134 <id>1</id>
135 <name> MOHAMED </name>
136 <posts>
137 <post>
138 <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
139 <topics>
1
2
3
4
PROBLEMS 35
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main> .\xml_editor.exe verify -i .\fixed_input7.xml -o fixed_input6.xml
[EH] There are 0 errors.
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main> .\xml_editor.exe verify -i .\input_file.xml -o fixed_input6.xml
[EH] Missing opening tag for </id> at line 126
[EH] Missing closing tag for <name> opened at line 126
[EH] Missing closing tag for <id> opened at line 129
[EH] Missing closing tag for <followers> opened at line 124
[EH] There are 4 errors.
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main>
powershell
powershell


input_file.xml
fixed_input6.xml
outputerrorHandling.xml
ErrorHandling.cpp
ErrorHandling2.cpp
XML_to_graph.cpp
main.cpp
Generate
Simulate
fixed_input6.xml
<users>
</user>
<user>
<id>6</id>
<name> MOHAsMED </name>
<posts>
<post>
<body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
<topics>
<topic>economy</topic>
<topic>finance</topic>
</topics>
</post>
<post>
<body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqu
<topics>
<topic>solar_energy</topic>
</topics>
</post>
</posts>
<followers>
<follower>
<name><id>1</id>
</name></follower>
<follower>
<id>2
</id></follower>
</followers></user>
<user>
<id>1</id>
<name> MOHAMED </name>
<posts>
PROBLEMS 35
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main> .\xml_editor.exe verify -i .\fixed_input7.xml -o fixed_input6.xml
[EH] There are 0 errors.
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main> .\xml_editor.exe verify -i .\input_file.xml -o fixed_input6.xml
[EH] Missing opening tag for </id> at line 126
[EH] Missing closing tag for <name> opened at line 126
[EH] Missing closing tag for <id> opened at line 129
[EH] Missing closing tag for <followers> opened at line 124
[EH] There are 4 errors.
PS C:\Users\20112\Desktop\CSE331-Project-XML-Editor-main>
powershell
powershell

XML to graph:

- Files: [graphRepresentation/XML_to_graph.cpp]
- Transform: uses `xmlToGraphFromTree` to build `map<int,user>` from the `XmlNode` tree. ` emits DOT edges and `visualizeGraph` calls `dot` to render an image

Data model & representation:

- **Nodes:** users (unique `id`).
- **Edges:** follower relations; an edge u->v means `u` follows `v` (or depending on semantics, `followers` list contains incoming edges). Implementation stores users in a `map<int,user>` where each `user` holds a `vector<int> followers` (adjacency list-like structure).
- **Space:** adjacency lists (per-user vector) — memory O(V + E) where V = #users and E = #follower-entries.

Network Analysis:

- File: [network/network_analysis.cpp]

- purpose: uses `xmlToGraphFromTree` to build `map<int,user>` from the `XmlNode` tree.
  Then preform the required command from the information from the generated graph.

Most Influencer:

-compute max follower count by scanning each user and reading `followers.size()`, then adding every user with max follower count to a vector.

-Time O(V) where V is the number of vertices or the number of users

-space O(V) in worst case if they all have the same amount of followers they would be listed in the vector

```
vector<pair<string,int>> MostInfluencer(map<int,user> users) {
    int mx = 0;
    vector<pair<string,int>> result;
    for ( auto &p : users) {
         user &u = p.second;
        if (u.followers.size() > mx) mx = u.followers.size();
    }
    for ( auto &p : users) {
         user &u = p.second;
        if (u.followers.size() == mx) result.push_back({u.name, u.id});
    }
    return result;
}
```

Mutual users:

- given a list of input ids S, the implementation iterates users in S, marks who they follow, and counts occurrences per candidate — effectively computing intersection of followers.

-worst Time O(S*V) where S is the number of ids given and V number of vertices or total users so if given all the users ids and all of them follow each other it becomes O(V^2)

-space O(V) in worst case if they all follow all the users

```cpp
vector<pair<string,int>> MutualUsers(map<int,user> users,vector<string>v) {
    vector<pair<string,int>> Mutual_Users;
    map<int,int> mp; // count how many of the provided ids follow a given user

    for ( auto &p : users) {
         user &u = p.second;
        if (find(v.begin(), v.end(), to_string(u.id)) != v.end()) {
            for (int fid : u.followers) mp[fid]++;
        }
    }

    for ( auto &entry : mp) {
        if (entry.second == v.size()) {
            auto it = users.find(entry.first);
            if (it != users.end()) Mutual_Users.push_back({it->second.name, it->second.id});
        }
    }
    return Mutual_Users;}
```

Most Active:

-compute number of connection for each user by iterating through the followers of each user then take the max in a list

-Time O(V^2) where V is the number of vertices or the number of users

-space O(V) in worst case if they all have the same amount of followers they would be listed in the vector

SuggestFollowers (friend-of-friend):

-for a target user u, iterate u's followers F, then for each follower f in F iterate f's followers and suggest their followees not already followed by u

-Time O(V^2) where V is the number of vertices or the number of users

-space O(V) in worst case if they all have the same amount of followers they would be listed in the vector

Search

-File: [search/Search.cpp]

- Method: Iterate posts, tokenize text, case-insensitive match on word/topic, and return (postID, snippet).

- Algorithm: For search, traverse all post nodes and tokenize each post's body (split on whitespace and punctuation); normalize tokens (lowercase) and compare to the search term. Collect matching `(postID, snippet)` pairs. For topic search, compare topic strings directly. This is a linear scan; small optimizations include early exit per post or indexing if needed.

Complexity of every module:

- **Parsing:** time O(N) where N = number of characters/tokens (tokenize once and build tree in recursive pass). Memory O(N) (tree nodes).
- **Formatting/Minifying/JSON conversion:** all tree traversals, time O(M) where M = nodes (≈O(N)).
  Memory :O(depth) stack.
- **Search**: scanning posts is O(P * L) where P = number of posts, L = average length per post; practically O(N).
- **Graph building & analysis:** building adjacency lists O(V + E).
- **Compression:** if algorithm is Huffman-like: building frequency table O(N), encoding O(N); if RLE: O(N).

compression and per-function space complexity

- **`compress_helper`** — Time: O(n) expected (single pass); Space: O(n) for compressed output plus dictionary storage. In terms of input size `n`, peak memory is O(n + D * Lavg) where `D` is dictionary entries and `Lavg` is average stored sequence length; typically O(n).

- **`decompress_helper`** — Time: O(k + m) where `k` is number of codes and `m` is decompressed output length; Space: O(m + D * Lavg) (output buffer + dictionary), practically O(m).
- **`parse_xml`** — Space: O(n) for the input byte buffer (reads entire file into memory) and removes BOM if present.
- **`parse_comp`** — Space: O(k) for the vector of `unsigned short` codes read from file.
- **`save_compressed_file` / `save_decompressed_xml`** — These stream to disk; note **`save_decompressed_xml`** currently takes its vector by value and will make an extra O(m) copy unless changed to take a const reference.

Per-function space complexity (key modules)

- **`ReadXml(ifstream&)`:** O(n) (stores tokens for the entire file).
- **`parse_node(const vector<string>&, int&)`:** O(1) extra per call (recursion stack); overall tree allocation O(n).
- **`parseXMLFile(const string&)`:** O(n) tokens + O(n) tree = O(n) peak.
- **`FormatXMLFromFile(...)`:** O(depth) recursion stack; O(n) if producing in-memory output.
- **`minifying(...)`:** O(depth) stack; O(n) if storing output in memory.
- **`convertXMLtoJSONFromTree(...)`:** O(n) for JSON output.
- **`compress(..)`:** O(n) input buffer + O(dict) (practically O(n)).

- **`decompress(..)`:** O(m) decompressed output + O(dict); plus an extra O(m) when `save_decompressed_xml` copies the buffer.
- **`xmlToGraphFromTree(XmlNode*)`:** O(V + P) where V is number of users and P total posts/auxiliary data.
- **`dotFileInput(...)`:** O(1) extra (streams edges to file) beyond the `users` map.
- **`visualizeGraph(...)`:** O(1) (invokes external `dot`).
- **`SearchByWordFromTree(...)` / `SearchByTopicFromTree(...)`:** O(R) for results plus traversal stack; overall O(n) if many matches.
- **`error_handling(...)`:** Implementation-dependent; could be O(n) if loading file into memory or O(1) if streaming corrections.

GUI

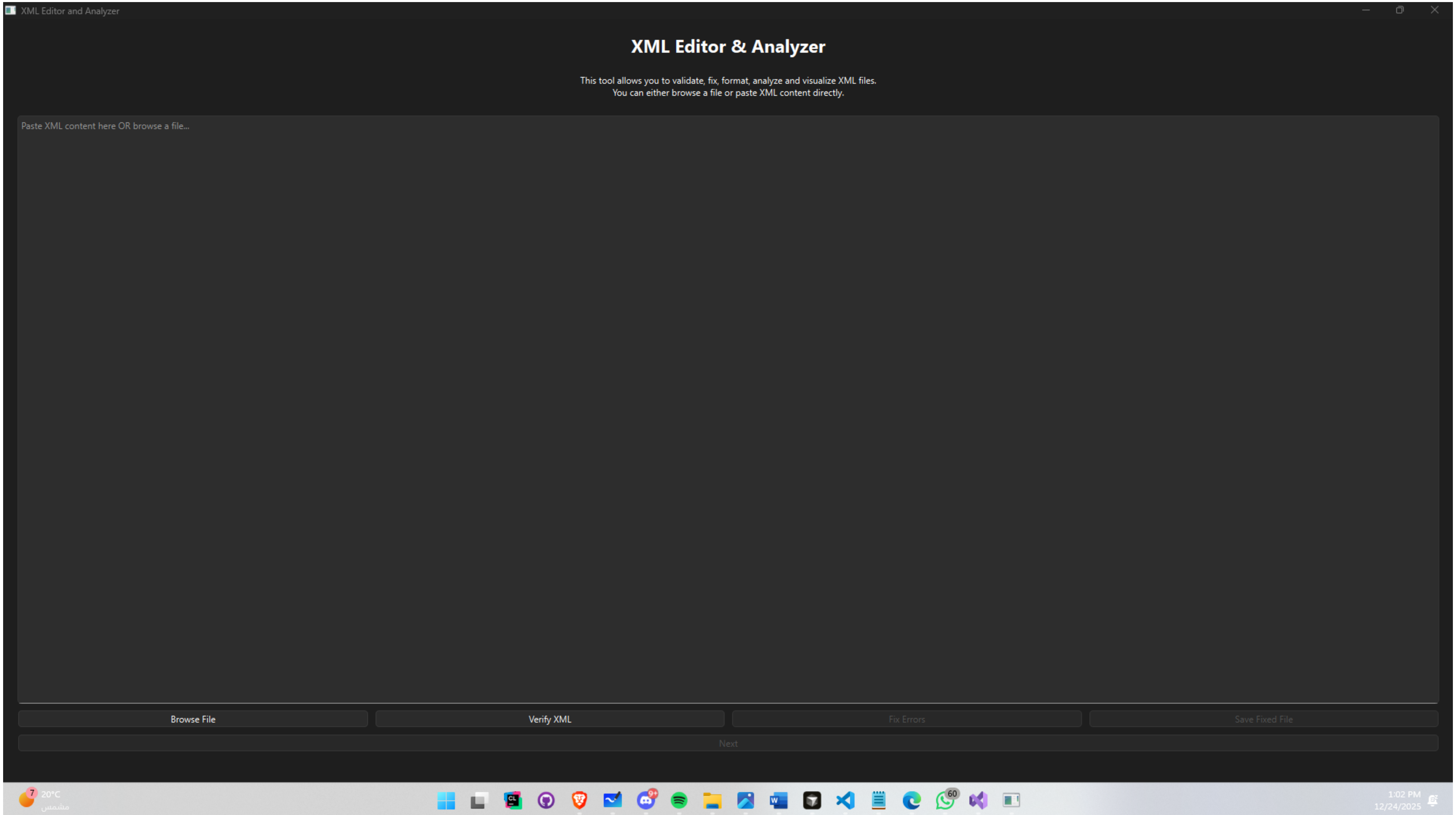
Overview: A lightweight Qt-based GUI provides file-open/save, command buttons (Format, Minify, JSON, Compress/Decompress, Verify, Draw), and output previews for formatted XML, JSON, compressed files, and graph images.

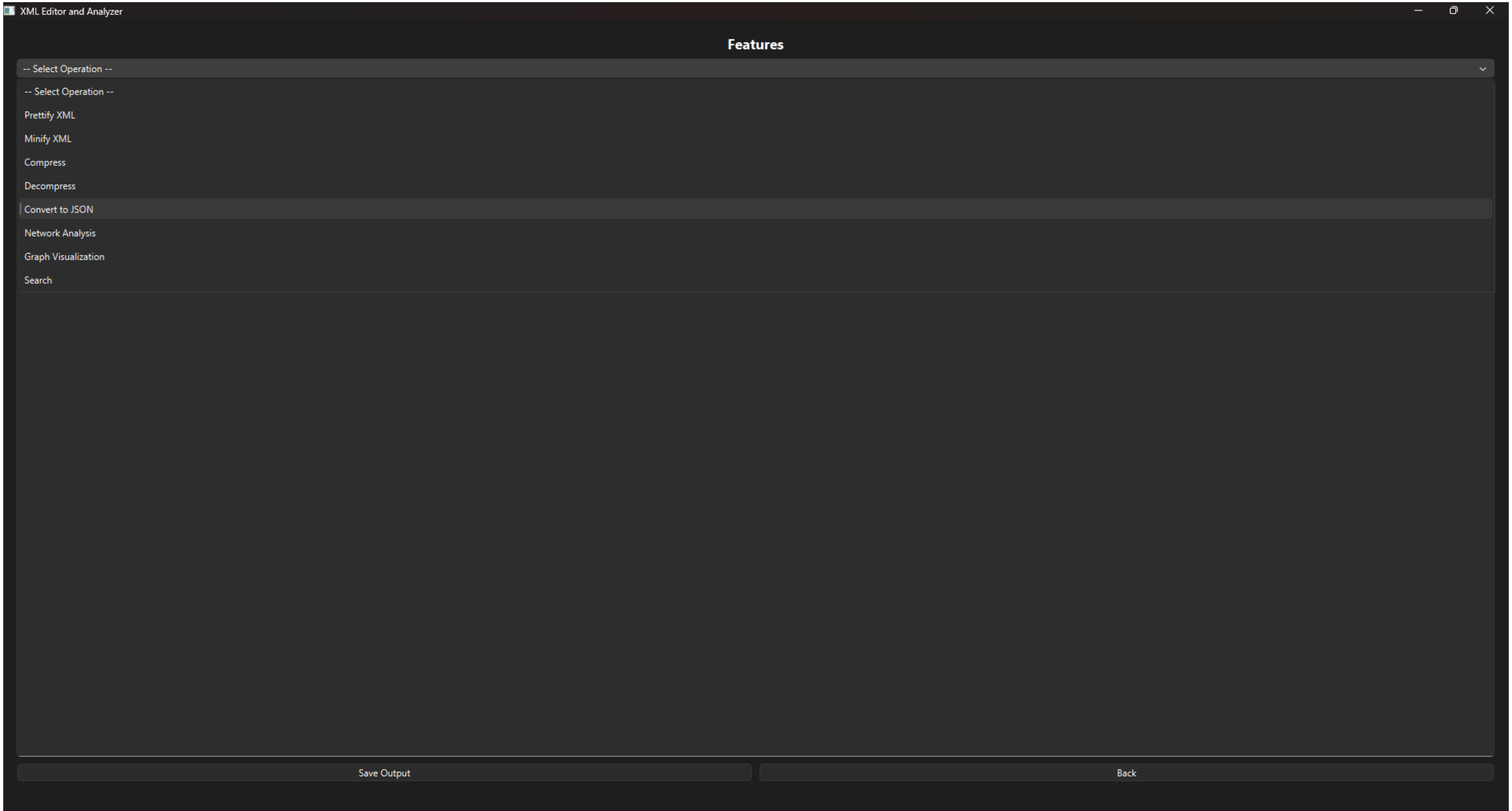
Implementation:

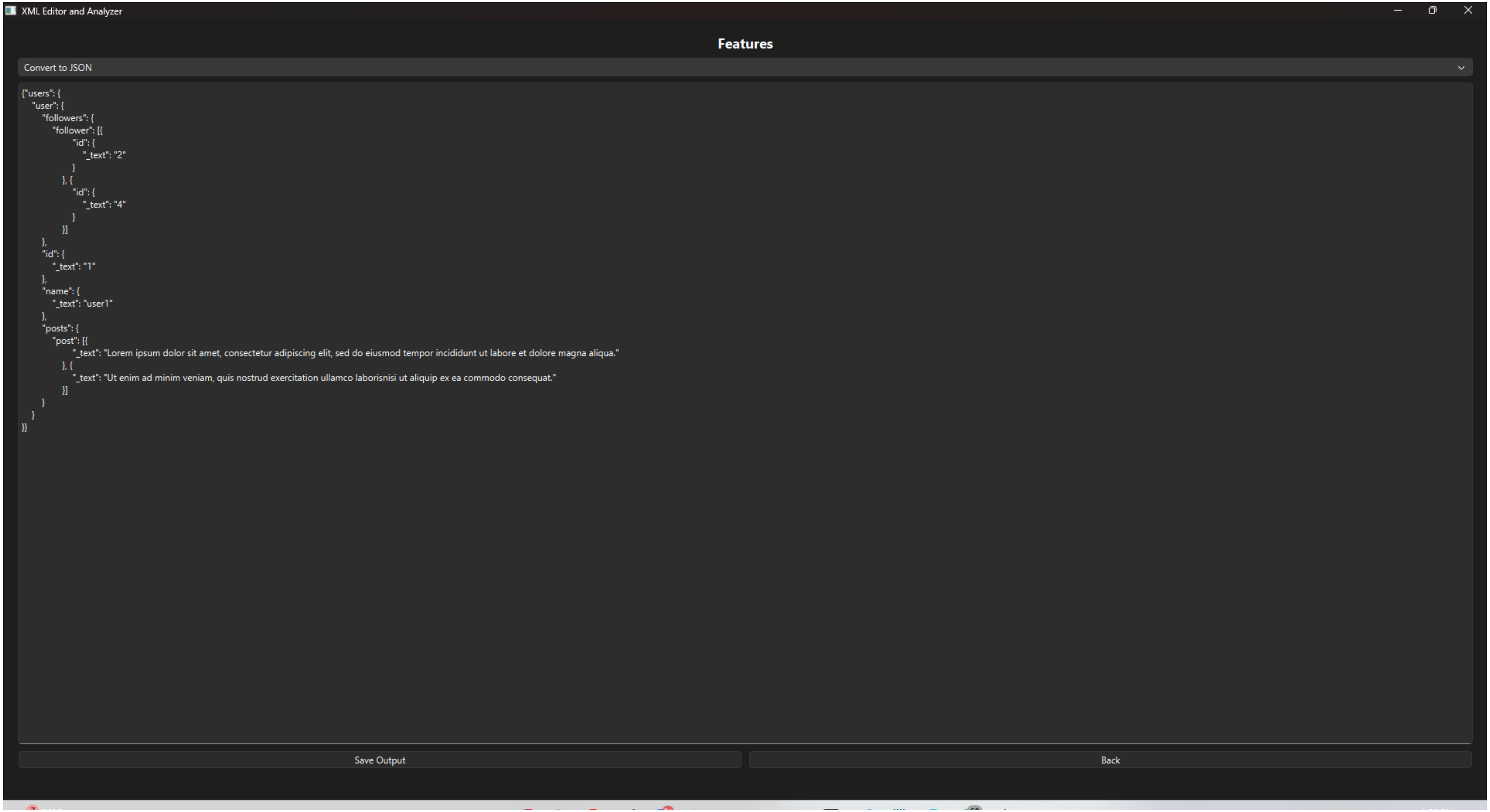
A desktop UI built with Qt Widgets that exposes the main CLI features (prettify, minify, compress/decompress, convert to JSON, network analysis, graph visualization, search). The top-left control is a dropdown to select the active operation; a feature list presents available operations; the central area is a large preview pane showing command output (example: JSON output); the bottom row contains action buttons (`Save Output`, `Back`). The UI uses a dark theme and large scrollable text area for previews .

Layout:

- Operation selector: `QComboBox`.
- Feature list / actions: `QListWidget` or `QTreeWidget` for grouped operations.
- Output preview: `QTextEdit` configured as read-only for output preview and editable when user chooses to edit output.
- Image preview (graph): `QLabel` or `QGraphicsView` to show Graphviz-generated JPG/SVG.
- Buttons: `QPushButton` for `Save Output`, `Back` and other actions.
- Layout: top toolbar / combo row, main horizontal splitter with left operations and right preview, status bar at bottom.


XML Editor and Analyzer
XML Editor & Analyzer
This tool allows you to validate, fix, format, analyze and visualize XML files.
You can either browse a file or paste XML content directly.
Paste XML content here OR browse a file...
Browse File
Verify XML
Fix Errors
Save Fixed File
Next
20°C
1:02 PM
12/24/2025


XML Editor and Analyzer
Features
-- Select Operation --
-- Select Operation --
Prettify XML
Minify XML
Compress
Decompress
Convert to JSON
Network Analysis
Graph Visualization
Search
Save Output
Back


XML Editor and Analyzer
Features
Convert to JSON
{"users": {
"user": {
"followers": {
"follower": [{
"id": {
"_text": "2"
}
}, {
"id": {
"_text": "4"
}
}]
},
"id": {
"_text": "1"
},
"name": {
"_text": "user1"
},
"posts": {
"post": [{
"_text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."
}, {
"_text": "Ut enim ad minim veniam, quis nostrud exercitation ullamco laborisnisi ut aliquip ex ea commodo consequat."
}]
}
}
}}
Save Output
Back

User flow (example): Convert to JSON)

1. User selects `Convert to JSON` from the dropdown.
2. User picks an input XML via `Open` or types a path, then clicks a run button.
3. GUI runs `xml_editor.exe json -i <in> -o <out>` (or calls the conversion function) via `QProcess`.
4. When process produces output, the QTextEdit preview is updated with pretty-printed JSON; `Save Output` becomes enabled to write the preview to disk.

Refrences:

- W3C: Extensible Markup Language (XML) 1.0 Specification.
- RFC 8259 — The JSON Data Interchange Format.
- D. A. Huffman, "A method for the construction of minimum-redundancy codes." 1952.
- cppreference.com — C++ standard library references.
- Graphviz documentation (https://graphviz.org)
- Qt documentation (`https://doc.qt.io`) for `QProcess`, `QFileDialog`, `QThread`, and UI components.

-Team Contributions

Farah Haitham Saddik Abd Elmaged	XML to graph + report
Mohammed Yasser Said	Minify XML +XML to json+ report
Karim Hosam Ahmed Ali	GUI
Maya ahmed farahat	ERROR handeling
Omar Abdelgaber Elsayed	Compress and decompress
Mohamed Ashraf Mohamed Mounir	Main + XML to tree parsing + search
Radwa Yasser Ahmed	Network analysis and formating