CSE331s: Data Structures and algorithms

Xml Project Milestone 1

Team Members

Name	Id	section
Beshoy Ashraf Faheem	1805453	2
Engy Fayek Adeeb	1806935	4
Kirollos Ashraf Sedky	1807624	3
Susanna Michael Faheem	1809355	2
Kirollos Medhat Massoud	1809488	3

GitHub Repo Link

https://github.com/kirollosashrafsedky/data-structure-xml-project

Video Link

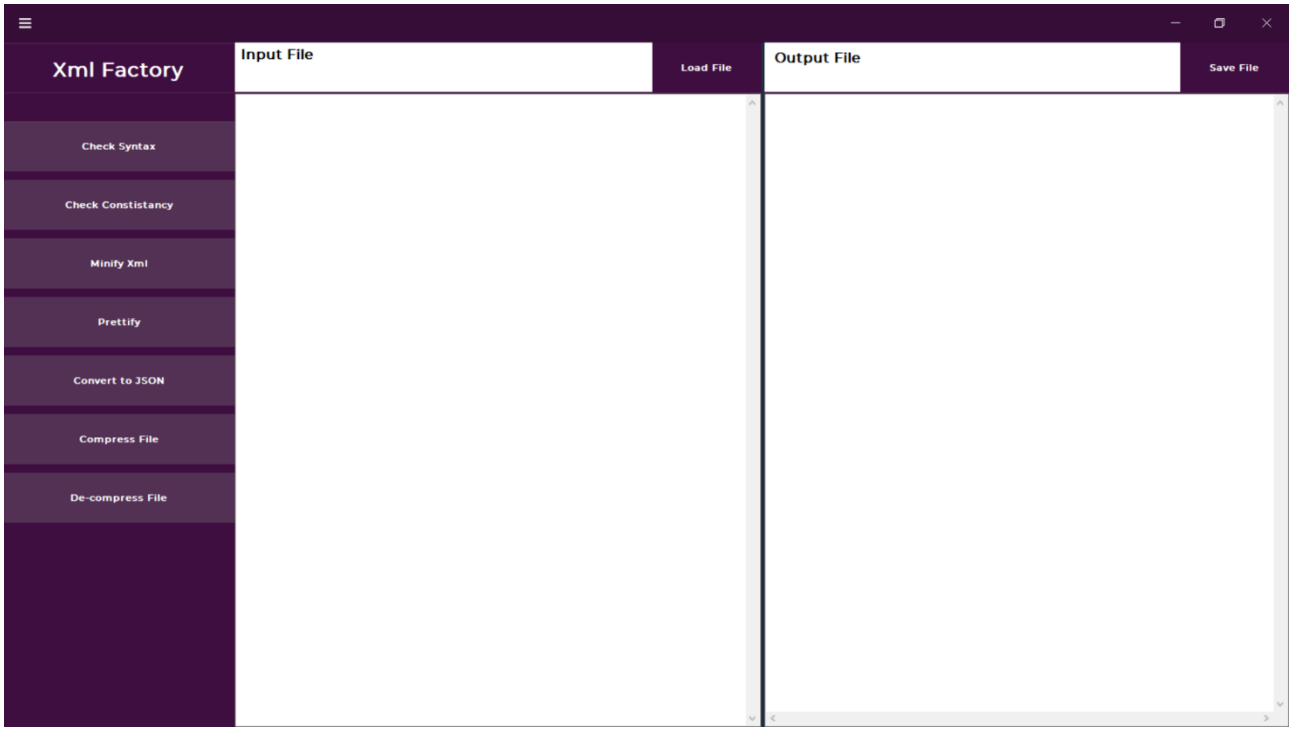
https://drive.google.com/drive/u/0/folders/1YkRrme4L-7cAHQxBbX4T2blJeBh7O33L

GUI

We used Visual Studio for the GUI using CLR (Common Language Runtime).

The window is split into 4 sections

1- Top bar, custom top bar that contains 4 buttons, Close, Maximize, Minimize, and Menu toggler button, the show/hide the menu
2- Menu Section to the left that contains the buttons of the different features of the program
3- Input file section that contains the name of the input file, it's content and Load file button that open file explorer to choose a file Available input extensions are **.xml** => an extension for xml files, And **.cxml,** an extension for compressed xml files "Will be mentioned later"
4- output file section that contains the output file after various operations, and Save File button to save a file, available save extensions are **.xml** for xml files, and **.cxml** for compressed files.



Project Description:

A complete xml GUI project that has the following features:

1- Load an xml file with extension ".xml"
2- Load a compressed xml file with extension ".cxml"
3- Check syntax of the input xml ('<' and '>' brackets), and outputs the error found along with its line and column
4- Check consistency of xml tags and correct them to output a valid xml file, it outputs the error found along with its line and column
5- Minify the input xml file
6- Prettify the input xml file
7- Convert any xml file to json " prettified and valid json, that has the ability to detect similar xml tags in the same level with the same names and put them in an array in json format, as repeated object names are not valid in json unlike xml"
8- Compress the xml file using Huffman's tree method, it was tested using a big file of about 643 KB and the output was 428KB, about 66% of the original file, The efficiency increases by increasing the file size and by using repeated characters. The compressed xml file is saved in a new extension ".cxml"

9- Decompressing the output ".cxml" file to obtain the original file again.
10- The ability to save the output file of all the previous operations in one of the following appropriate extensions ".json, .xml, .cxml"

List of Classes

1- **XmlError** => A class for xml errors to contain the error msg string, number of line, and col.
Members, int row, int col, string errorMsg
Methods, getRow, getCol, getErrorMsg, setRow, setCol, setErrorMsg

2- **XmlAttribute** => A class for xml attributes that contain only the property and value for every attribute.
Members, string property, string value
Methods, getProperty, setProperty, getValue, setValue

3- **XmlTreeNode** => A class that represents a tree node for the xml tag, every tag can contain list of attributes, list of children nodes, a value, a tag name and have a parent
Members, string tagName, string value, XmlTreeNode* prent, vector <XmlTreeNode*> children, vector<xmlAttribute*> attributes
Methods, getTagName, setTagName, getValue, setValue, getParent, setParent, getChildren, appendChild, popLastChild, getAttributes, appendAttribute, popLastAttribute, hasChildren, getNumberOfChildren
Important methods descriptions:
    1- **void appendChild(XmlTreeNode* child)**
    pushes a child to the vector of childs and sets his parent to the current node, with time complexity: O(1) and space of O(1)
    2- **void appendAttribute(XmlAttribute* attribute)**
    pushed a new attribute to the vector of the node attributes, with time complexity: O(1) and space of O(1)

3- **bool hasChildren() const**
returns true if the node has children, 0 otherwise, with time complexity: O(1) and space of O(1)

4- **XmlDocument** => A class that contains the root of the xml tree, it points to the first node in the tree,
Members, XmlTreeNode* root
Methods, getRoot, setRoot, appendChild, appendAttribute, toString
Important methods descriptions:
1- **static void toString(XmlTreeNode* node, std::string& output, std::string indentationTabs = "");**
recursive function of time complexity O(N), each node is visited only once, it traverse the xml tree and calls itself for each node, For each one it prints the prettified node with its tag name, value, attributes and then call itself again for its children and so on,
This function is used in the prettify function, it prints the whole xml with tabs for each levels as it stores the indentation level. Once the xml is converted to a tree, this function is called on this tree which outputs the prettified xml.

5- **JsonNode:** A class that contain the data for a single node in json. It's an abstract class, two classes inherit from it which are **JsonFinalNode**, and **JsonCompoundNode**. It's similar to the xmlNodeTree class except its for json, and it's abstract because json has more than one type of nodes which have many things in common
Members, string nodeName, JsonNodeType nodeType, JsonNode* parent "Note: JsonNodeType is an enum that contains { **JsonFinalNodeType**, **JsonObjectNodeType**, **JsonArrayNodeType** }
Methods, getNodeName, setNodeName, getNodeType, setNodeType, getParent, setParent, toString

6- **JsonFinalNode:** A class that inherit from **JsonNode** class, it stores the data of a final node that don't have any other children "for example {"prop": "value"}.
Members: string nodeText
Methods: getNodeText, setNodeText, toString + the inherited methods and members of JsonNode class

Important methods descriptions:

1- **std::string toString(std::string indentationTabs) const** function that returns a string of the node prop and value with complexity of O(1) as it doesn't have any other children nodes. Ex output: "prop": "value"

7- **JsonCompundNode:** A class that inherit from **JsonNode** class, it stores the data of a node that can have children like objects or arrays, ex of objects: { "prop1": "value1", "prop2": "value2"}
Ex of array nodes, ["value1", { "prop1": "value1"}]. A compound node can also have children of other compound nodes.
Members: vector<JsonNode*> children
Methods: getNodeArray, setNodeArray, appendChild, toString, searchChildren, deleteChild + the inherited methods and members of JsonNode class

Important methods descriptions:

1- **std::string toString(std::string indentationTabs) const** recursive function of time complexity O(N), each node is visited only once, it traverse the json tree and calls itself for each node, For each one it prints the prettified node with its node name, and then call itself again for its children and so on, for object nodes it prints out { , } and in between calls itself for nested nodes, For arrays, the same as objects with [,] instead of { , }. And it stops when reaching the final node in the tree.
the xml attributes are converted to json property, value pair, also the text of the xml is converted to #text prop if the xml tag has other nodes.

2- **void seachChildren(string nodeName, JsonNode** parentNode)**
linear search function with complexity O(N) that returns true if a node exists in a specific parent with the given nodeName.

3- **Void deleteChild(JsonNode* child)**
Time complexity O(N), it searches for a node and deletes it

8- **Json:** A class that is similar to XmlDocument class it points to the first element in the json tree.
Members: JsonCompoundNode* jsonHeadNode
Methods: getHeadNode, setHeadNode, appendChild, toString
These methods are very simple they just call other methods of the JsonNodeTree class mentioned above with the same complexity

General important functions

These functions uses the previous class to do the features listed above, the are not present in one file. Each group of functions related to a specific function are present in a separate file

1- bool checkXmlSyntax(const std::string& xml, XmlError& error)
param 1 => xml string
param 2 => xml error object
return => true if correct, false otherwise.
Time Complexity => O(N)
Space Complexity => O(1)
Function Description => it uses a stack to store the opening and closing brackets <, > to check their consistency, this is not tags consistency check but it's a required step before calling the check consistency function.

2- bool checkXmlConsistancy(const std::string& xmlString, std::string& correctXml, XmlError& error)

param 1 => xml string

param 2 => correct xml, input as empty string

param 3 => empty error object to fill if the function found any errors, not used if the xml is consistant

return => true if no erros found, false otherwise.

Time Complexity => O(N)

Space Complexity => O(N), it stores the new xml in correctXml

Function Description => it uses a stack to store the opening and closing tags the tag word itself to check their consistency, it also solves the error by appending the tags found in the stack with </ > then append it to the correct xml file. Note that the function generates valid xml syntax but not necessarily a correct semantics tag.

3- void xmlParse(const std::string& xmlString, XmlDocument& xmlDoc)

param 1 => xml string

param 2 => empty object of xmlDocument type to fill with the tree

return => void

Time Complexity => O(N) because it calls generateXmlTree func which is recursive with complexity O(N)

Space Complexity => O(N), it stores the tree in xmlDoc

Function Description => It calls the function generateXmlTree initially which is a recursive function used to build the tree,

Limits => it requires valid xml

4- void generateXmlTree(const std::string& xmlString, int& i, XmlTreeNode* xmlNode)

param 1 => xml string

param 2 => index of the current char in the xml string

param 3 => pointer to an xml node to access it and it's children

return => void

Time Complexity => O(N) because calls itself recursively each time it sees a new tag and make a node for that tag, then call itself again for the children tags and so on
Space Complexity => O(N), it stores nodes in heap proportional to the number of tags in the string
Function Description => It builds the xml tree, it supports also attributes, xml values and children
Limits => it requires valid xml

5- std::string getStringTill(const std::string& str, int& index, char stopChar1, char stopChar2)
param 1 => string to save till a certain charachter
param 2 => index to start saving from
param 3 => first stop character
param 4 => second stop char, its used as an alternative for the first char to stop saving char, with ana initial value of '\0'
return => string
Time Complexity => O(N) where n is the length of the string starting from "index" till reaching the stop char 1 or 2
Space Complexity => O(1)
Function Description => it extracts the string till a specified character, it's a helping function used in building the xml tree to get tag names, attribute names and values and tag inner text

6- std::string removeSpaces(std::string& str)
param 1 => string remove spaces from
return => string
Time Complexity => O(N)
Space Complexity => O(N)
Function Description => removes spaces from start and end of the string, using the isspace function to detect space characters. It's a helping function used in the minification function

7- std::string minifcation (const std::string& xml)
param 1 => string to minify
return => minified string

Time Complexity => O(N)
Space Complexity => O(N)
Function Description => it's responsible for removing spaces from the xml tags before and after it, while keeping spaces between letters as they are because removing spaces between text may result in loss of data

8- void generateJsonTree(const XmlTreeNode* xmlNode, JsonNode* jsonNode)
param 1 => xml node to start from
param 2 => json node to build from the xml tree node
return => void
Time Complexity => best, average => O(N)
Worst case => O(N^2) in case of repeated xml tags in the same level
Space Complexity => O(N), a new node for each xml tag
Function Description => it's responsible for creating the json tree by traversing the xml tree node by node recursively and creating json nodes . Every attribute in the xml tag is generated as property value pair in the json object, if the xml tag has text and children as well, the text is written in this form "#text": "sample text"

9- void xmlToJson (const XmlDocument& xml)
param 1 => xml document that containthe xml tree to create th json node for
return => void
Time Complexity => the same as the previous function because it calls it internally
Space Complexity => the same as the previous function because it calls it internally
Function Description => it's gets the xml tree as an input, generates the json tree, then simply calls the function toString from the json class that returns the json prettified.

10- char strToChar(const std::string& str, int index)
param 1 => sequence string of zeros and ones to convert to char to be saved

param 2 => index to start reading 8 chars from
return => charachter
Time Complexity => O(1), it contains a loop with fixed size (8)
Space Complexity => O(1)
Function Description => it gets a sequence of zeros and ones and start reading 8 chars starting from index then convert them to char to be saved, it's used in the compression and decompression

11- Std::string toBinary(unsigned char n)
param 1 => char to convert to sequence of zeros and ones
param 2 => index to start reading 8 chars from
return => string which is a sequence of zeros and ones of a certain character
Time Complexity => O(1) , it contains a loop with fixed size (8)
Space Complexity => O(1)
Function Description => it gets a char and convert it to a sequence of zeros and ones string, used in compression and decompression

12- Node* getNode(char ch, int freq, Node* left, Node* right)
param 1 => character
param 2 => its frequency
param 3 => pointer to left node
param 4 => pointer to right node
return => pointer to the created node
Time Complexity => O(1)
Space Complexity => O(1)
Function Description => it creates a node with the given values. It's a helping function used in building the Huffman's tree

13- void encode(Node* root, std::string str, std::map<char, std::string>& huffmanCode)
param 1 => root node
param 2 => string to store the Huffman code
param 3 => map of Huffman codes
return => void

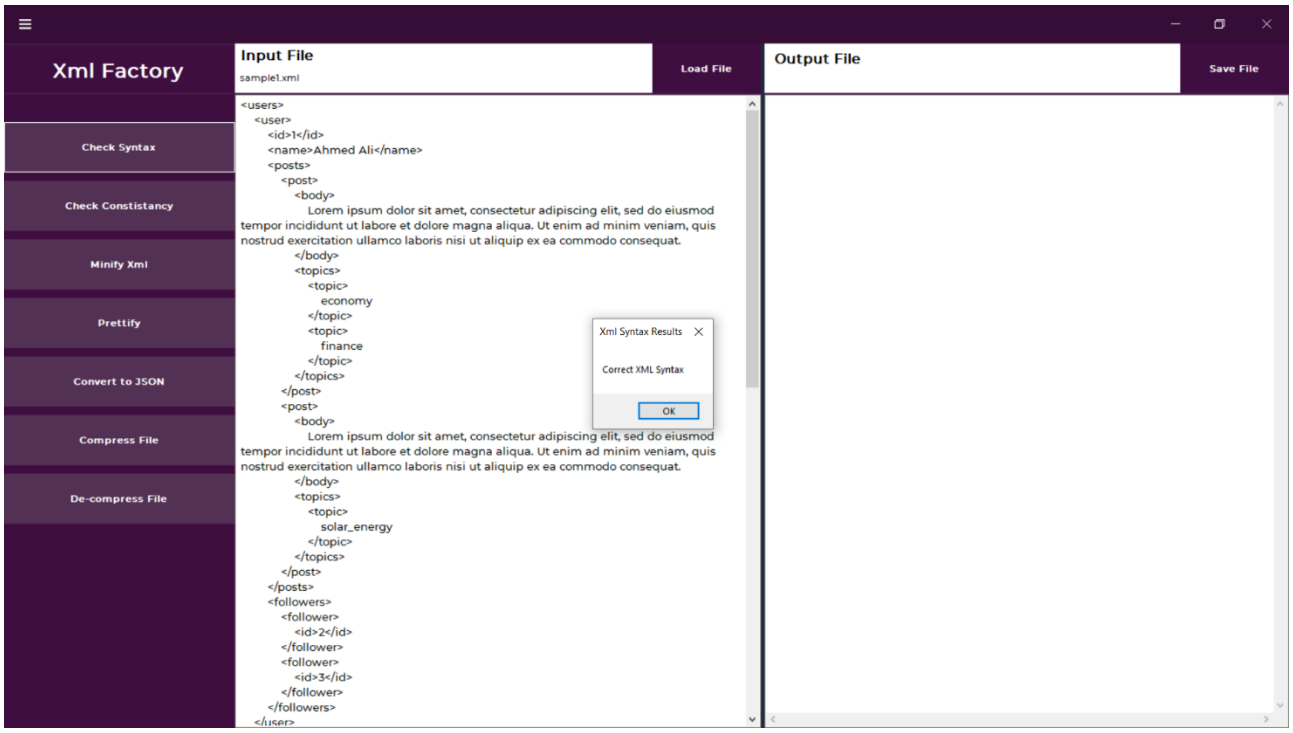
Time Complexity => O(n)
Space Complexity => O(1)
Function Description => it creates the huffman's tree using recursion.

14- std::string compressXml(std::string text)
param 1 => text to compress
return => compressed string
Time Complexity => O(n^2)
Space Complexity => O(1)
Function Description => calculates the frequency of each character in a linear way then uses a priority queue to create nodes and store them then call the above encode function

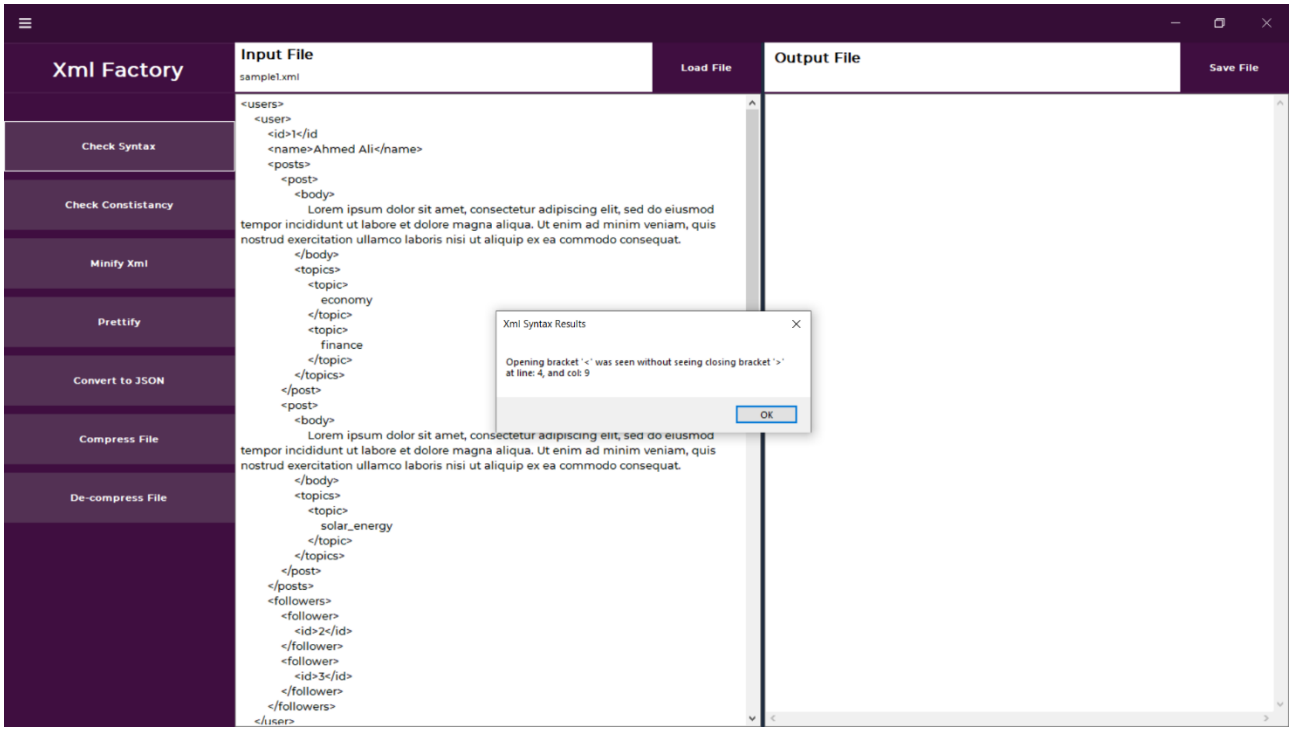
15- std::string decompressXml(std::string text)
param 1 => text to decompress
return => decompressed string
Time Complexity => O(n)
Space Complexity => O(1)
Function Description => it extract the map from the string chars and create the huffmancode tree again then when it meets a special string which is "&&", it starts to decode the string again to its original form

Screenshots from the output

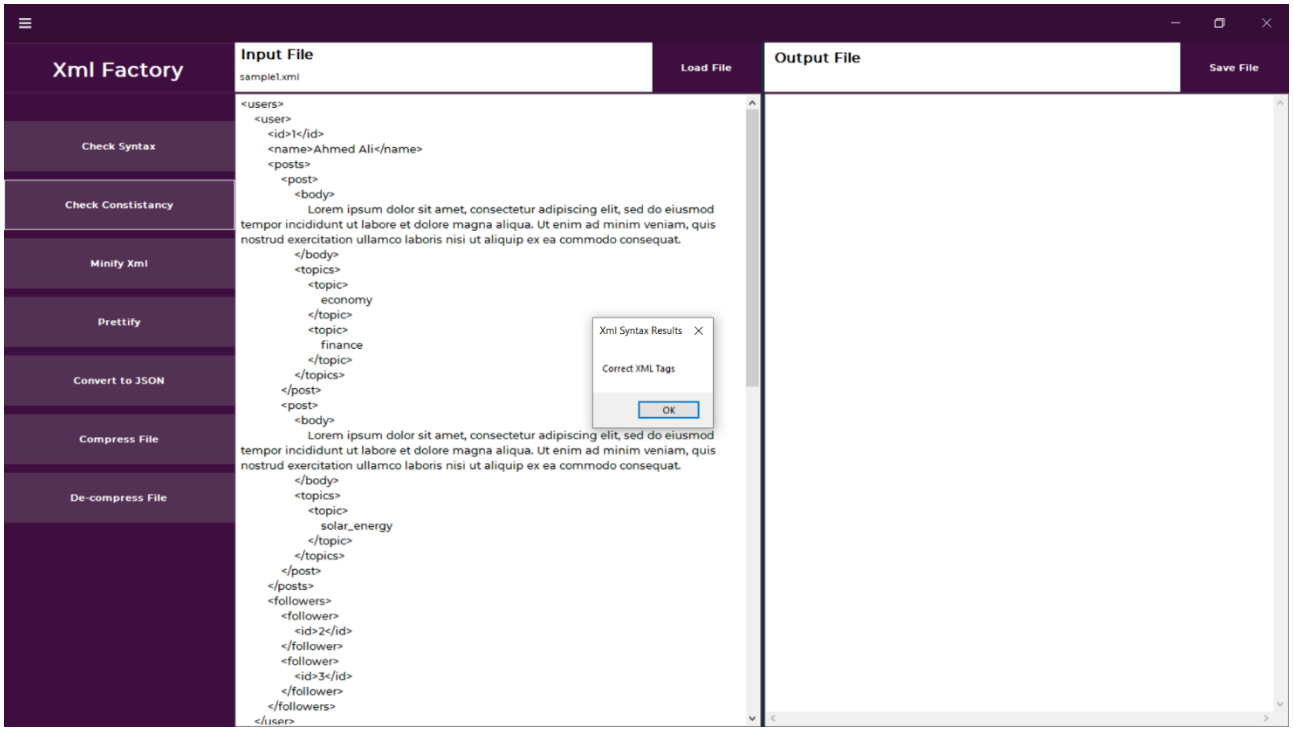
- Correct xml Syntax



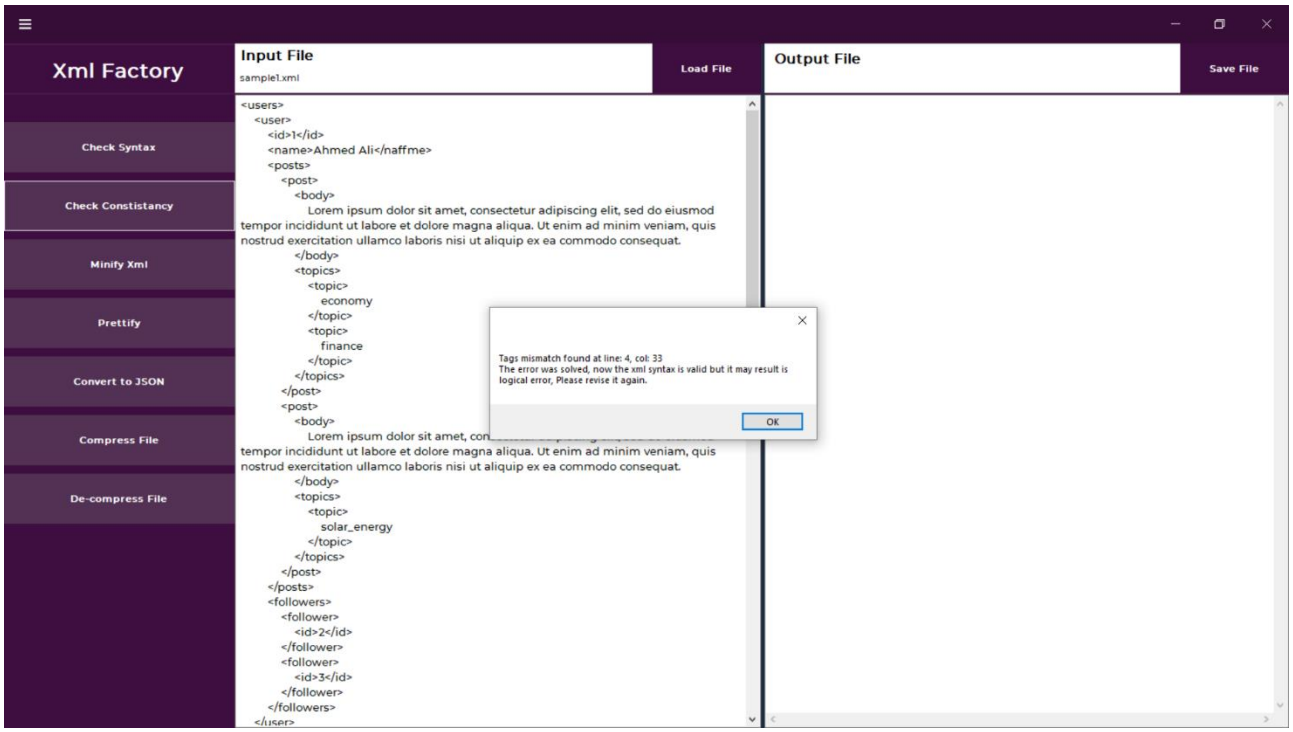
- Wrong xml syntax



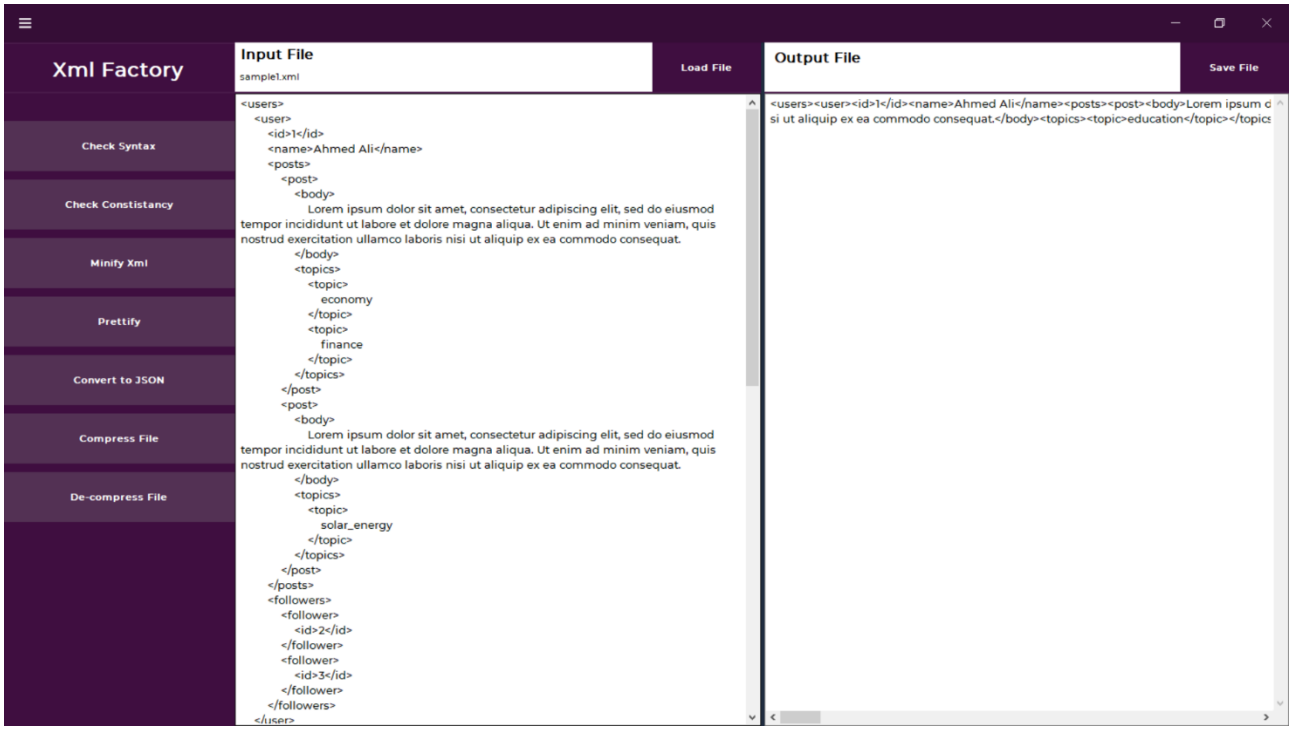
- Consistent xml tags



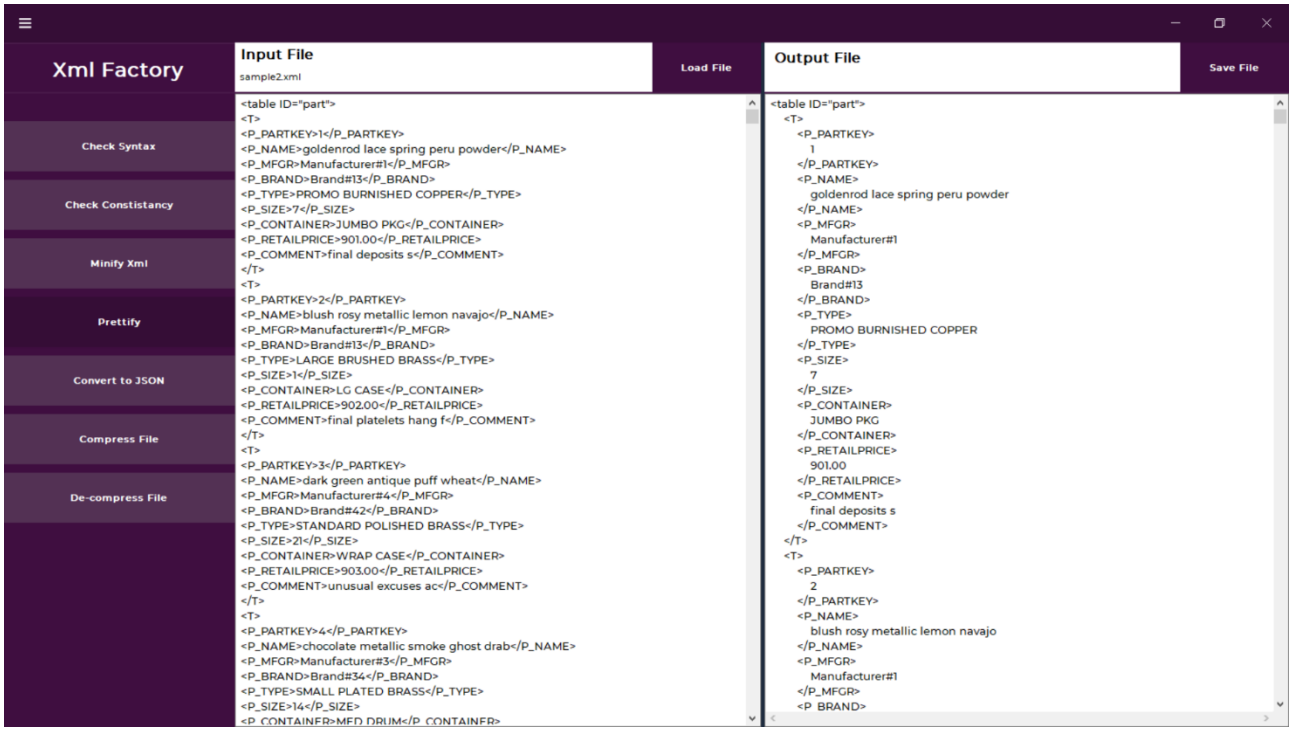
- Inconsistent xml tags



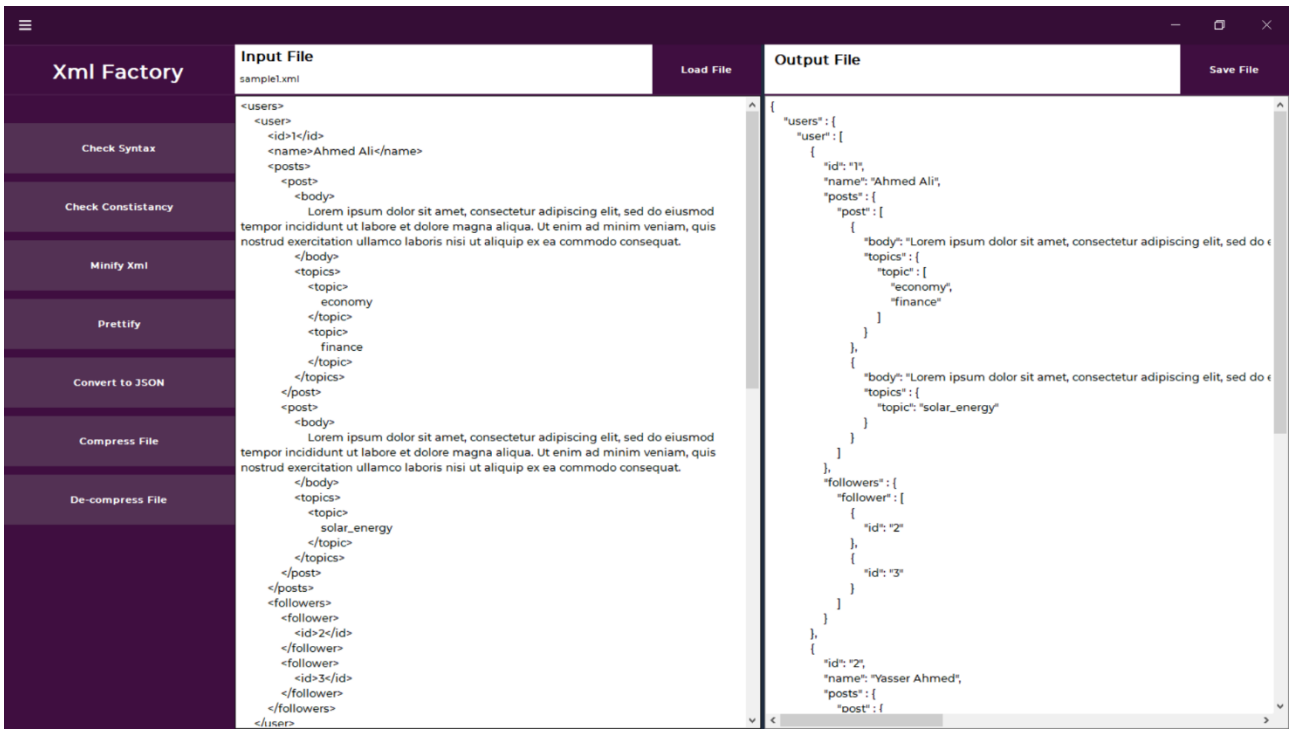
- Minification



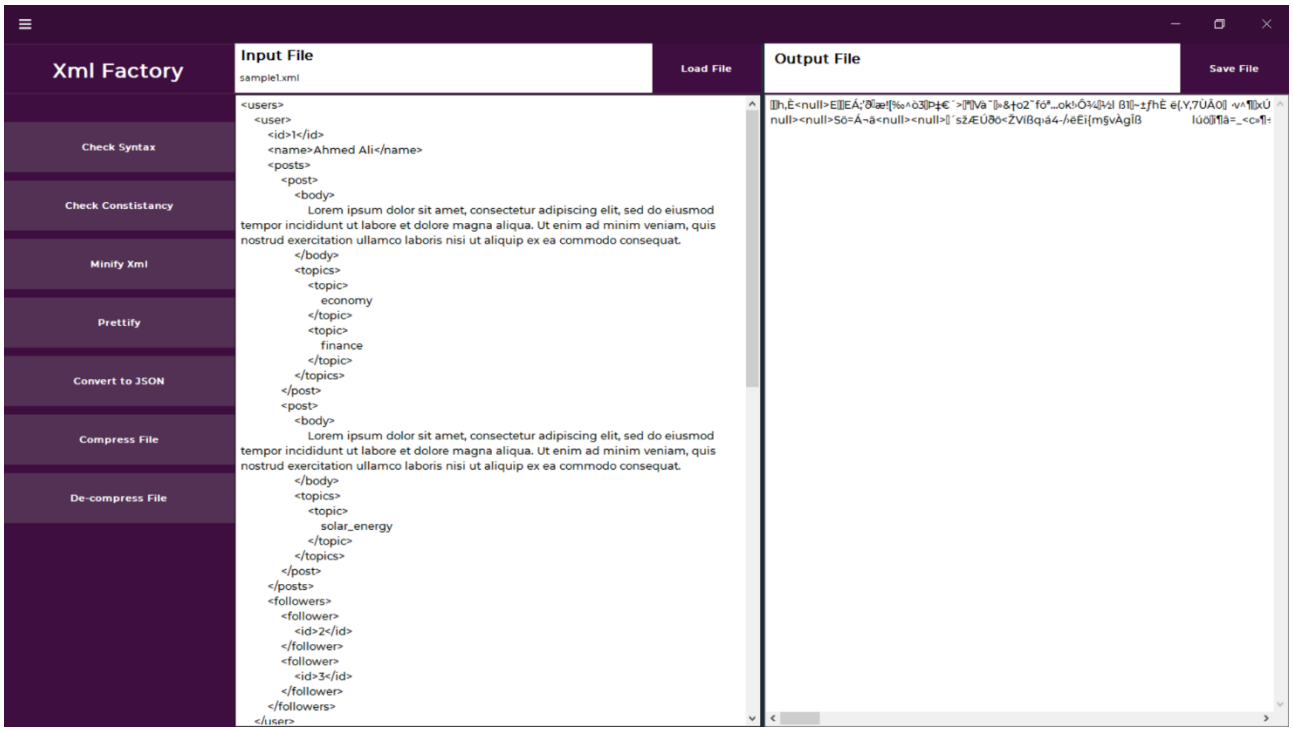
- Prettifying



- Convert to JSON



- Compression



- De-compression