JSON VALIDATOR

Name	ID
PHOLL CHANPHAYUK	e20220554
SOK KIMHENG	e20221601
TEK CHANSETHA	e20220186
SARITH SEYLA	e20221116

Lecturer: MR. OL Phearun

TABLE OF CONTENTS

Objectives

Our objective is to create a website where we can let users input text or upload file to verify if the input is a valid JSON file or not. Our validation included:

- Unicode: UTF-8 encoded
- Whitespace: Allowed but in insignificant amount (except in strings)
- Tokens: ***[***, ***]***, ***{***, ***}***, **:**, **,**, ***strings***, ***numbers***, ***true***, ***false***, ***null***.
- Strings: Must be in double-quoted, backslash before special or control characters.
- Case-sensitive: true, false, null (must be lowercase).

METHODS & LANGUAGES:

JFLEX+JAVA

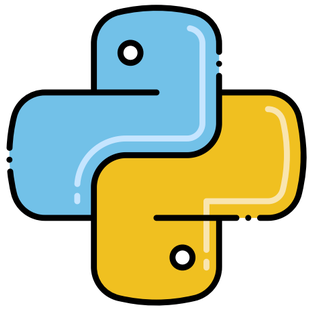
Is used for JSON lexer and our own custom JSON parser.

HTML, CSS, AND JAVASCRIPT

Used for GUI, it includes text box for user input, button to upload file to analyze or the user can simply drag and drop the file into our website to analyze instead.

PYTHON

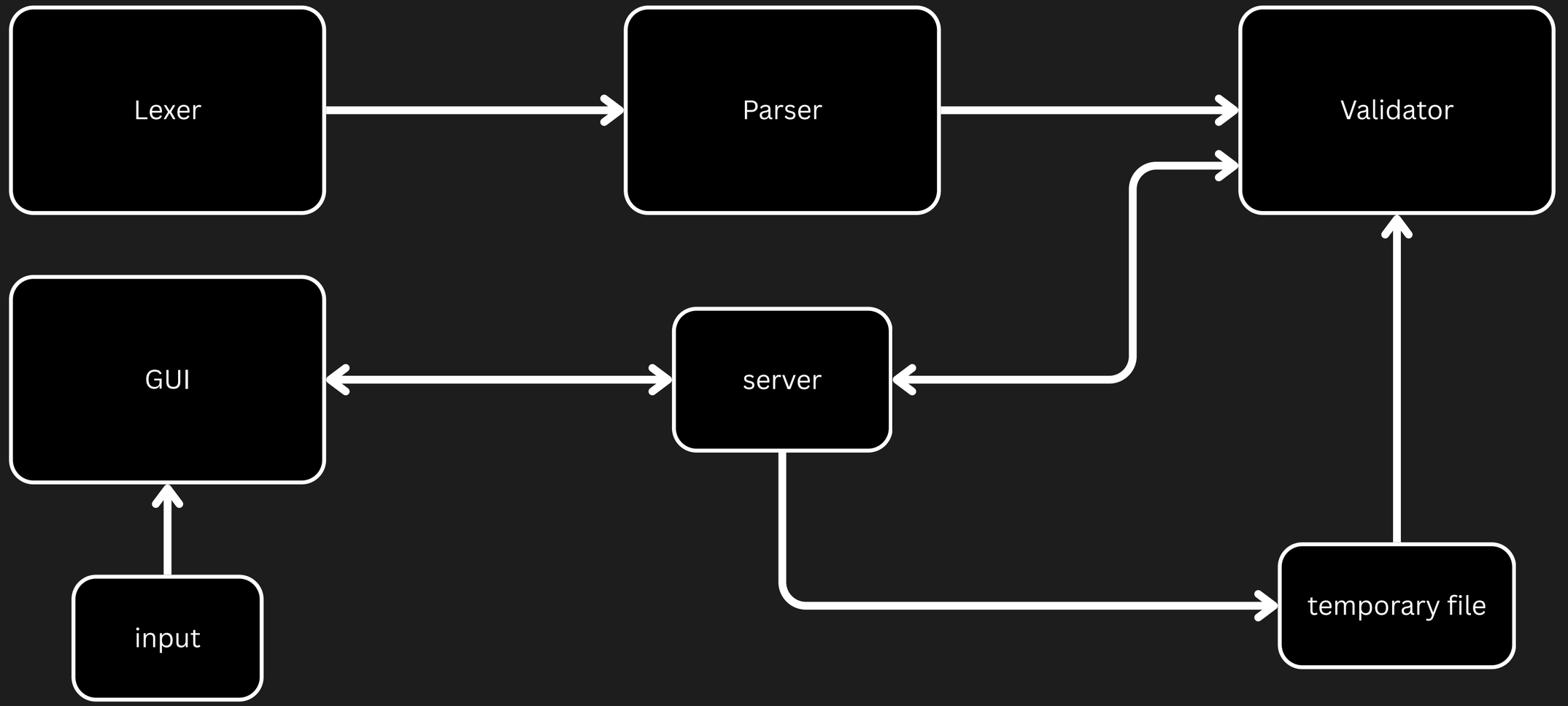
It is used for hosting the website and for our backend operations behind the scene.



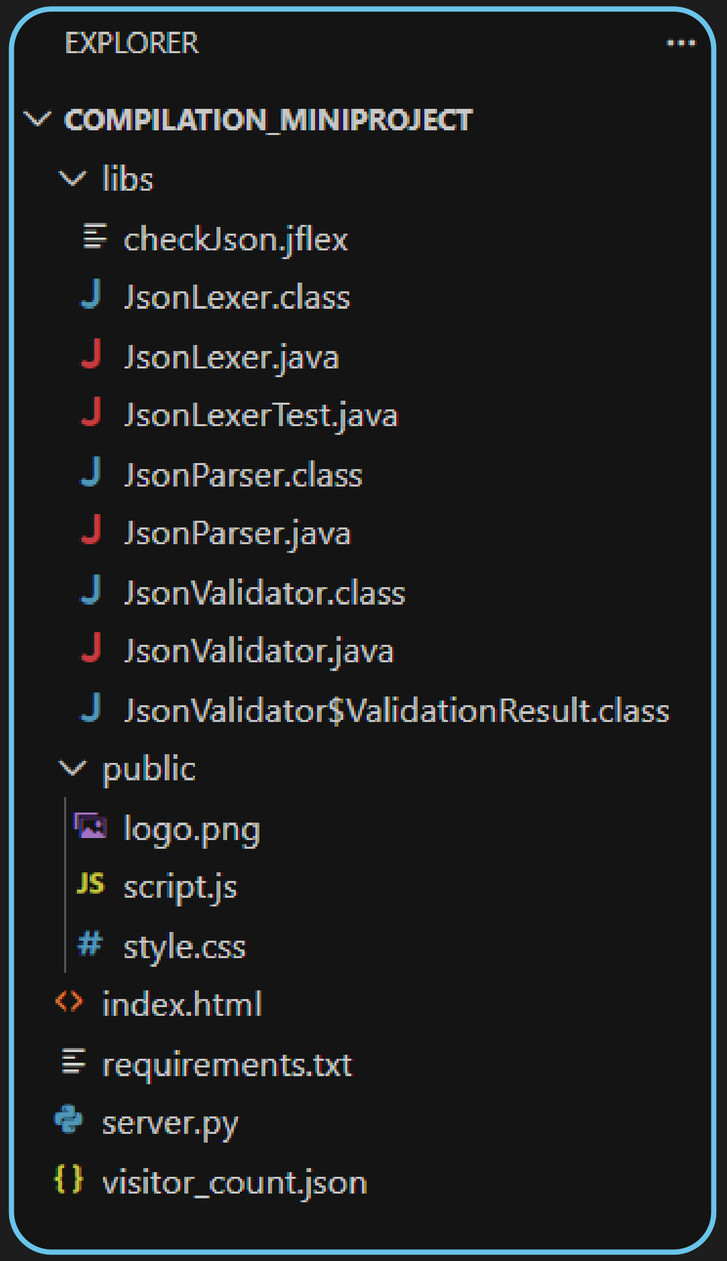
Implementaion

https://bit.ly/team_A19

Flow



Project Folder



checkJson.jflex

```
%%

%class JsonLexer
%unicode
%line
%column
%type String

%{
    public static final String LEFT_BRACE = "LEFT_BRACE";
    public static final String RIGHT_BRACE = "RIGHT_BRACE";
    public static final String LEFT_BRACKET = "LEFT_BRACKET";
    public static final String RIGHT_BRACKET = "RIGHT_BRACKET";
    public static final String COMMA = "COMMA";
    public static final String COLON = "COLON";
    public static final String STRING = "STRING";
    public static final String NUMBER = "NUMBER";
    public static final String TRUE = "TRUE";
    public static final String FALSE = "FALSE";
    public static final String NULL = "NULL";
    public static final String ERROR = "ERROR";
    public static final String EOF = "EOF";

    private void error(String message) {
        System.err.println("Error at line " + (yyline + 1) + ", column " + (yycolumn + 1) + ": " + message);
    }
%}
```

checkJson.jflex

```
WhiteSpace = [ \t\r\n]+

Digit = [0-9]
NonZeroDigit = [1-9]
Integer = "-"? ("0" | {NonZeroDigit}{Digit}*)
Fraction = "."{Digit}+
Exponent = [eE][+-]?{Digit}+
Number = {Integer}{Fraction}?{Exponent}?

InvalidNumber = "-"? "0"{Digit}+ | {Number}"."{Digit}*"."
StringChar = [\u1780-\u17FF\u19E0-\u19FF]
ValidEscape = "\\" ([\"\\/bfnrt] | "u"[0-9a-fA-F]{4})
InvalidEscape = "\\" [^\"\\\/bfnrtu] | "\\" "u" [0-9a-fA-F]{0,3}[^0-9a-fA-F] | "\\" "u" [0-9a-fA-F]{0,3}
String = \" ({StringChar} | {ValidEscape})* \"

UnterminatedString = \" ({StringChar} | {ValidEscape})*
InvalidEscapeString = \" ({StringChar} | {ValidEscape})* {InvalidEscape}
ControlCharString = \" ({StringChar} | {ValidEscape})* [\u0000-\u001F]
InvalidTrue = [Tt][Rr][Uu][Ee]
InvalidFalse = [Ff][Aa][Ll][Ss][Ee]
InvalidNull = [Nn][Uu][Ll][Ll]

%%
```

checkJson.jflex

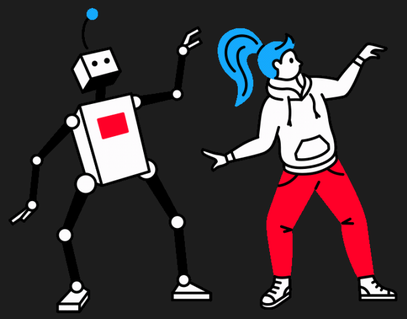
```
/* Lexical rules  */
<YYINITIAL> {
    "{"                 { return LEFT_BRACE; }
    "}"                 { return RIGHT_BRACE; }
    "["                 { return LEFT_BRACKET; }
    "]"                 { return RIGHT_BRACKET; }
    ","                 { return COMMA; }
    ":"                 { return COLON; }

    "true"              { return TRUE; }
    "false"             { return FALSE; }
    "null"              { return NULL; }

    {InvalidTrue}       {
                          error("Invalid boolean: '" + yytext() + "' (must be lowercase 'true')");
                          return ERROR;
                        }
    {InvalidFalse}      {
                          error("Invalid boolean: '" + yytext() + "' (must be lowercase 'false')");
                          return ERROR;
                        }
    {InvalidNull}       {
                          error("Invalid null: '" + yytext() + "' (must be lowercase 'null')");
                          return ERROR;
                        }
    {InvalidNumber}     {
                          error("Invalid number format: '" + yytext() + "'");
                          return ERROR;
                        }

    {Number}            { return NUMBER; }
    {InvalidEscapeString} {
                          error("Invalid escape sequence in string");
                          return ERROR;
                        }
    {ControlCharString} {
                          error("Unescaped control character in string");
                          return ERROR;
                        }
    {UnterminatedString} {
                          error("Unterminated string");
                          return ERROR;
                        }
    {String}            { return STRING; }
    {WhiteSpace}        { /* ignore */ }

    <<EOF>>             { return EOF; }

    "'"                 {
                          error("Invalid quote character (use double quotes for JSON strings)");
                          return ERROR;
                        }
    .                   {
                          error("Unexpected character: '" + yytext() + "'");
                          return ERROR;
                        }
}
```

JsonParser.java

```java
private void parseValue() throws IOException {
    if (currentToken.equals(JsonLexer.ERROR)) {
        throw new RuntimeException("Lexical error encountered");
    }

    switch (currentToken) {
        case JsonLexer.LEFT_BRACE:
            parseObject();
            break;
        case JsonLexer.LEFT_BRACKET:
            parseArray();
            break;
        case JsonLexer.STRING:
        case JsonLexer.NUMBER:
        case JsonLexer.TRUE:
        case JsonLexer.FALSE:
        case JsonLexer.NULL:
            advance(); // Consume the literal
            break;
        default:
            error("Expected value but found " + currentToken);
            throw new RuntimeException("Parse error");
    }
}
```

```java
private void parseObject() throws IOException {
    expect(JsonLexer.LEFT_BRACE);

    // Empty object is valid
    if (match(JsonLexer.RIGHT_BRACE)) {
        return;
    }

    // Parse key-value pairs
    do {
        // Expect string key
        if (!currentToken.equals(JsonLexer.STRING)) {
            error("Expected string key but found " + currentToken);
            throw new RuntimeException("Parse error");
        }
        advance();

        // Expect colon
        expect(JsonLexer.COLON);

        // Parse value
        parseValue();

    } while (match(JsonLexer.COMMA));

    // Expect closing brace
    expect(JsonLexer.RIGHT_BRACE);
}
```

```java
/**
 * Parse JSON array: [ value, ... ]
 */
private void parseArray() throws IOException {
    expect(JsonLexer.LEFT_BRACKET);

    // Empty array is valid
    if (match(JsonLexer.RIGHT_BRACKET)) {
        return;
    }

    // Parse values
    do {
        parseValue();
    } while (match(JsonLexer.COMMA));

    expect(JsonLexer.RIGHT_BRACKET);
}
```



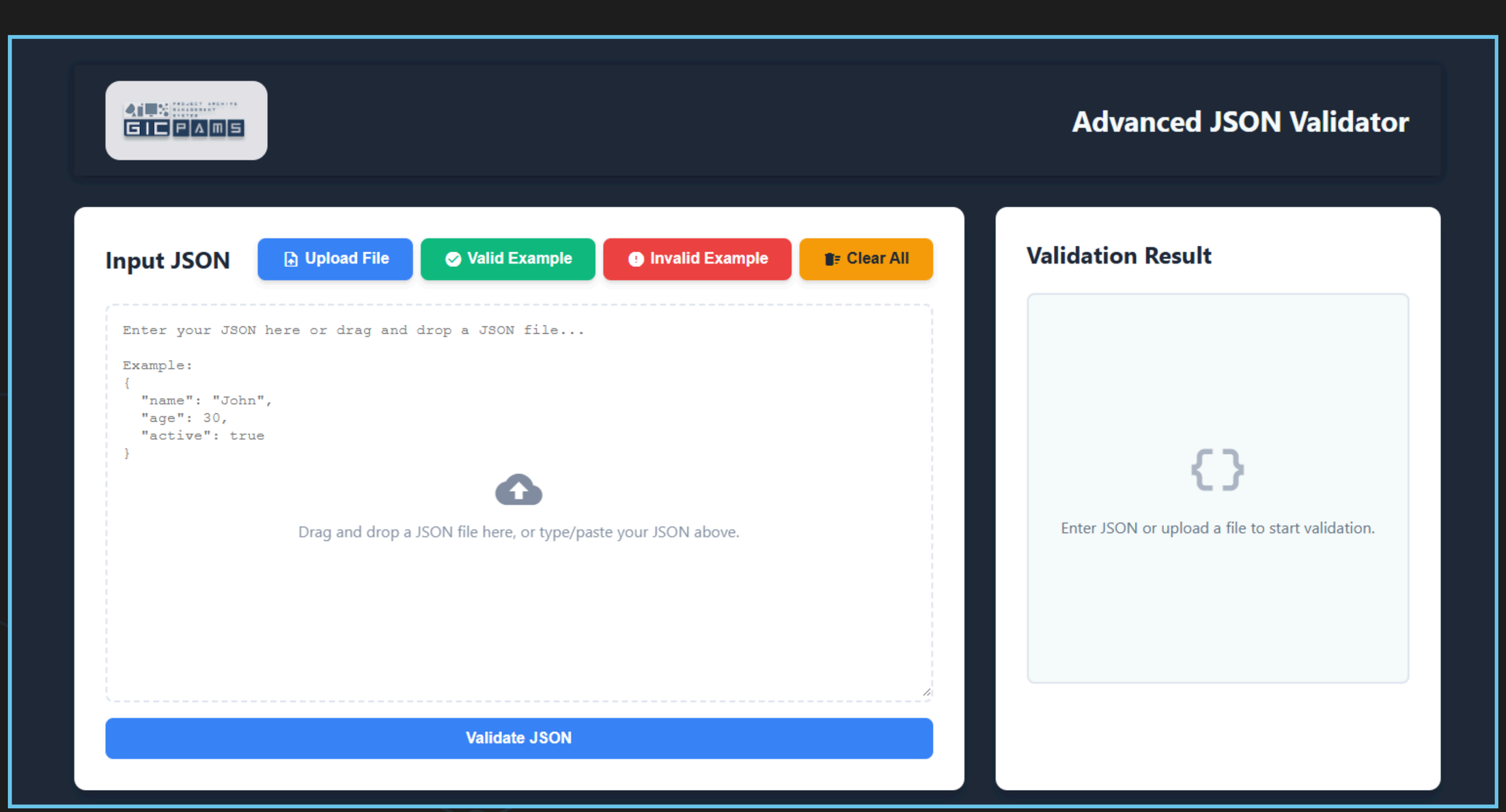
JsonValidator.java

```java
import java.io.*;

public class JsonValidator {

    public static class ValidationResult {
        private final boolean valid;
        private final String message;
        private final String[] errors;

        public ValidationResult(boolean valid, String message, String[] errors) {
            this.valid = valid;
            this.message = message;
            this.errors = errors;
        }

        public boolean isValid() {
            return valid;
        }

        public String getMessage() {
            return message;
        }

        public String[] getErrors() {
            return errors;
        }

        @Override
        public String toString() {
            StringBuilder sb = new StringBuilder();
            sb.append("========================================\n");
            sb.append("           JSON VALIDATION RESULT\n");
            sb.append("========================================\n");

            if (valid) {
                sb.append("Status: VALID JSON\n");
                sb.append(message).append("\n");
            } else {
                sb.append("Status: INVALID JSON\n");
                sb.append(message).append("\n");

                if (errors != null && errors.length > 0) {
                    sb.append("\nErrors Found:\n");
                    for (int i = 0; i < errors.length; i++) {
                        sb.append("  ").append(i + 1).append(". ").append(errors[i]).append("\n");
                    }
                }
            }

            sb.append("========================================");
            return sb.toString();
        }
    }
```

```java
    public static ValidationResult validate(String jsonString) {
        if (jsonString == null || jsonString.trim().isEmpty()) {
            return new ValidationResult(false, "Empty or null input",
                new String[]{"JSON input cannot be empty"});
        }

        try {
            StringReader reader = new StringReader(jsonString);
            JsonParser parser = new JsonParser(reader);
            boolean isValid = parser.parse();

            if (isValid) {
                return new ValidationResult(true,
                    "JSON is well-formed and structurally valid", null);
            } else {
                String[] errorArray = parser.getErrors().toArray(new String[0]);
                return new ValidationResult(false,
                    "JSON validation failed", errorArray);
            }

        } catch (IOException e) {
            return new ValidationResult(false,
                "IO error during validation",
                new String[]{"IOException: " + e.getMessage()});
        } catch (Exception e) {
            return new ValidationResult(false,
                "Unexpected error during validation",
                new String[]{"Exception: " + e.getMessage()});
        }
    }

    public static ValidationResult validateFile(String filename) {
        try {
            StringBuilder content = new StringBuilder();
            BufferedReader reader = new BufferedReader(new FileReader(filename));
            String line;
            while ((line = reader.readLine()) != null) {
                content.append(line).append("\n");
            }
            reader.close();

            return validate(content.toString());

        } catch (FileNotFoundException e) {
            return new ValidationResult(false,
                "File not found",
                new String[]{"Cannot find file: " + filename});
        } catch (IOException e) {
            return new ValidationResult(false,
                "Error reading file",
                new String[]{"IOException: " + e.getMessage()});
        }
    }
```

Result

https://bit.ly/team_A19


Advanced JSON Validator
Input JSON
Upload File
Valid Example
Invalid Example
Clear All
Enter your JSON here or drag and drop a JSON file...
Example:
{
"name": "John",
"age": 30,
"active": true
}
Drag and drop a JSON file here, or type/paste your JSON above.
Validate JSON
Validation Result
Enter JSON or upload a file to start validation.

Validation Features

Lexical Analysis
Detects invalid characters, malformed strings, and number formats

Structure Validation
Checks object/array nesting, brackets, and syntax rules

Fast Processing
Uses JFlex lexer and recursive descent parser

Detailed Errors
Provides specific error messages for debugging

Special features of our Web App

Drag and Drop
Allows users to drag and drop JSON files for validation

User friendly
Clean and intuitive interface for easy navigation

Use On everything
Responsive design for all devices

Access Everywhere
Accessible from any device with an internet connection

ITC GROUP A

Documentation Source Code

Total Visits: Tracking Offline

Demo

https://bit.ly/team_A19

https://bit.ly/team_A19

SCAN ME

Free 1$/person