Report Generator

SE 6387 - Advanced Software Engineering Project

Team Members:

- **Jyothise Johny (JXJ190035)**
- **Vishakha Singh (VXS200068)**
- **Jeya Visshwak Jeyakumar (JXJ190055)**

Table of Contents:

Description of Project:

- **Problem Statement:** To negate the use of high level latex coding and converting the latex code into a PDF.

- **Goals**: The main aim of our project is to generate a PDF from a latex code, we divided our goal in to various subgoals:
  - Setting up a JSON parser
  - Initial set of unit test cases
  - Setting up a HashMap with the keys and values according to the pdf sent by the professor.
  - Generating a pdf for the initial latex paragraph and trying to scale it from there

- **Stakeholder:** The stakeholders for this project are students and faculty members of the university .

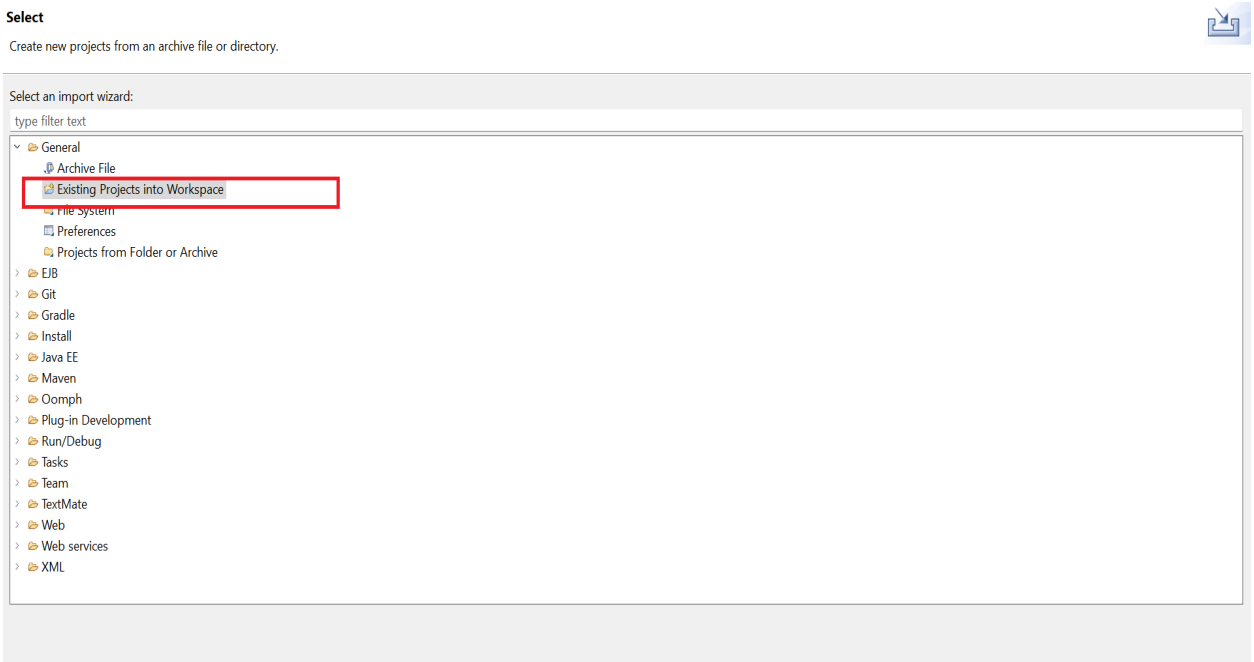
- **Technology Stack:** The technology stack being used is **Java** to code out the latex template and extract info from json, **MikTex** for latex compiler, **Junit** for test cases .

Project Setup:

- Clone the Project to your local and follow the below steps .

  Eclipse:

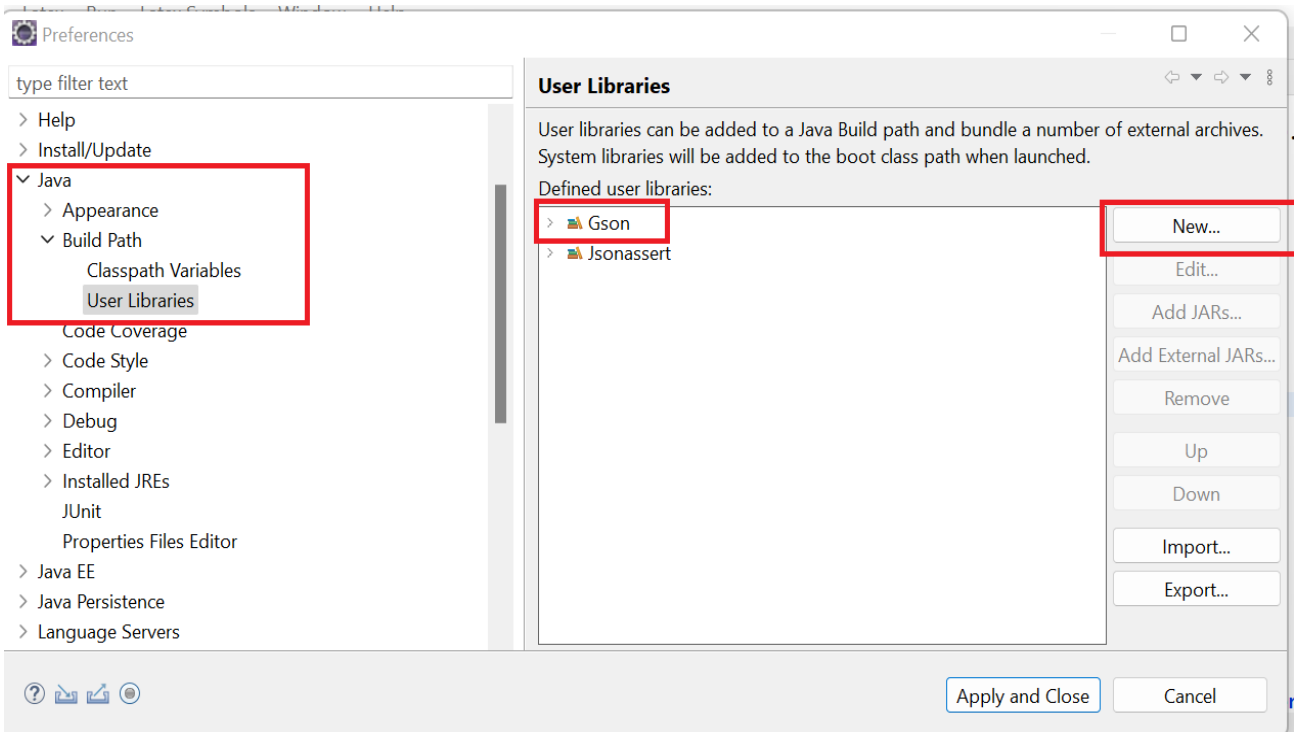
- Install Eclipse or any Java IDE .
- Configure the JDK path by going to Environment Variables. Eg : **JAVA_HOME = C:\Program Files\Java\jdk1.8.0_202**
- Once that's done, import the project that you cloned from git repo by following the below steps:
  - In Eclipse, **File -> Import -> General -> Existing Projects into Workspace.**

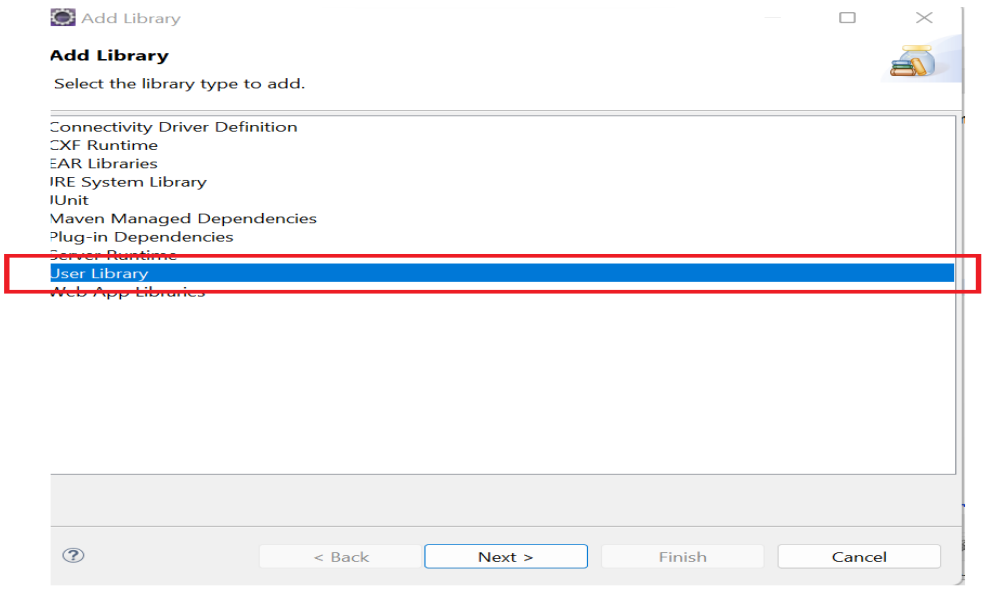


Json Parser for extracting json elements

- Install **gson.jar** file and follow the below steps to import the jar:
  - Go to Preferences in Eclipse.
  - Navigate to **Java -> Build Path -> User Libraries -> New** . Name the library .
  - After naming, click on **Add External Jars** and save .

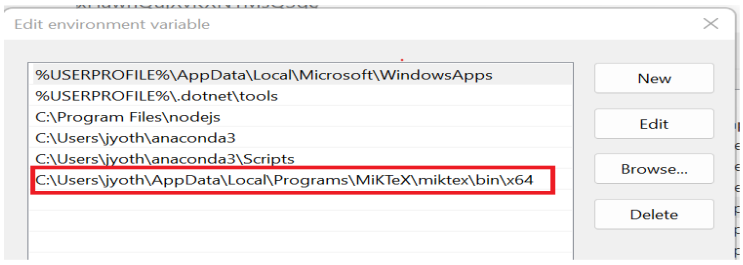
- Once the external jar is added, it needs to be added to the Project .
  - Right click on the Project.
  - Go to **Build Path -> Add Libraries -> User Library.**
  - Choose the User Library you created and save.



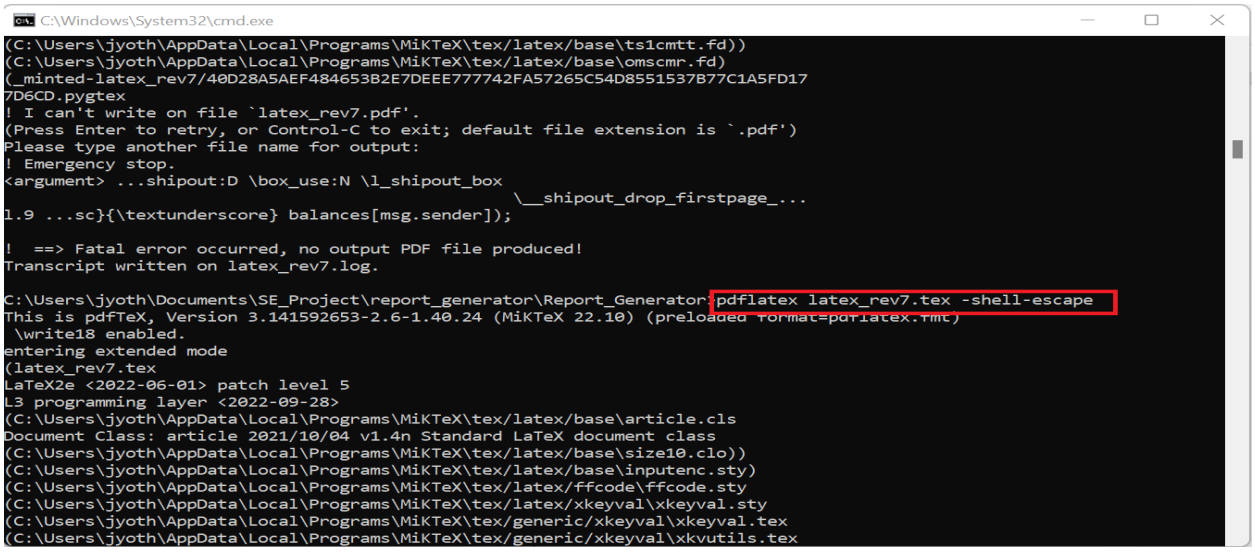


Latex Parser

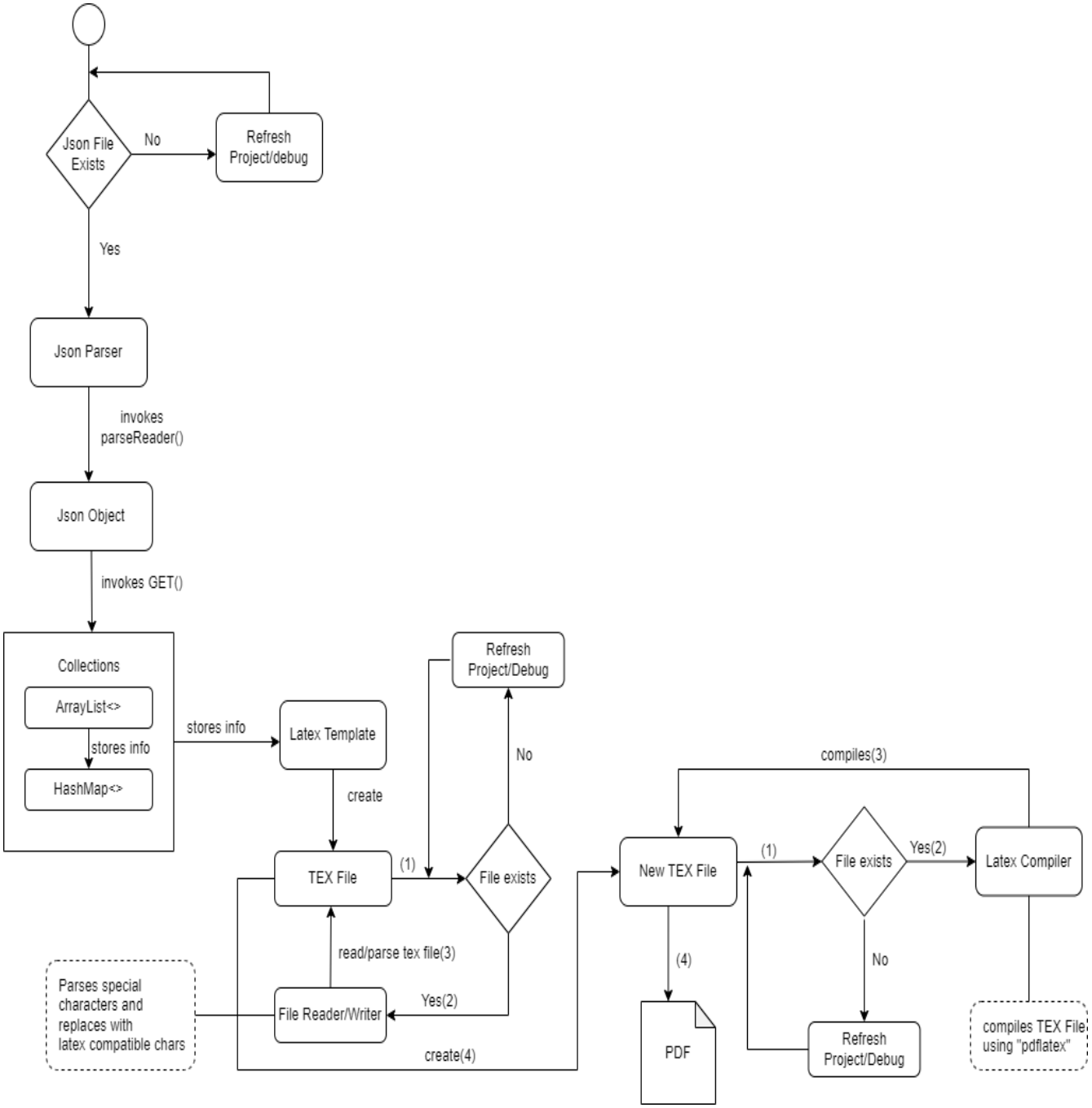
- Install **MikTex** into your system.
- Configure the path variable by navigating to Environment Variables .
    - Under **User Variable -> Path,** paste **C:\Users\jyoth\AppData\Local\Programs\MiKTeX\miktex\bin\x64.**
    - To test, open the command prompt and try running **pdflatex <filename.tex>.** The PDF will be generated in the project directory.



Once the Project has been imported and the above setup has been completed, make sure to configure the path to `elements.json` in your local system in the `Parser_Inner.java` and then run the `Parser_Inner.java` file. The generated latex file will be `latex_rev7.tex`. Open command prompt, run `pdflatex latex_rev7.tex -shell-escape`. The PDF file will be generated in your project directory itself.



High Level Diagram:



- Initially , we would be checking whether the Json file we would be extracting the data from exists or not. If it doesn't, then we stop . Else we continue to the next step.

- If the Json file exists, the json parser invokes the parseReader() method that parses the content and later calls the GET() method to retrieve the data elements from the file

- Once we retrieve the elements using the GET() method, we need to store them somewhere using Collections i.e **ArrayList**, **HashMap** etc . Here we use ArrayList<> to store multiple values associated with a single key and then a HashMap to map them to the key.

- Once the info from the json file is stored in HashMap, we need to write them into our hard coded latex template which will be later written into a TEX file using file reading/writing .

- There are few special characters which are latex compatible and won't be compiled until we make them compatible i.e '**\textunderscore**', '**\textendash**' etc. So each time we try to replace the characters, a new file object is being created which will be used to write the final content into a newly created TEX file.

- Once the special characters have been replaced, the TEX file is compiled using MikTex compiler from the command prompt with the help of "**pdflatex <filename> -shell-escape**" and the PDF is generated.

Code Snippets:

- Reading file and accessing nested elements

```
//Reading json file
JsonObject jsonObject = JsonParser.parseReader(new
FileReader("C:\\Users\\jyoth\\Documents\\SE_Project\\report_generator\
\Report_Generator\\src\\report\\generator\\jsonparser\\elements.json")
)
.getAsJsonObject();

//Initializing list to store values associated with a single key.
List<Object> inner_elements = null;

//Initializing map to map values associated with a single key.
HashMap<String, List<Object>> jsonMap = new HashMap<>();


JsonObject data = (JsonObject) jsonObject.get("data");
JsonObject item = (JsonObject) data.get("item");
JsonObject bug_type = (JsonObject) item.get("bug_type");
```

- Accessing key and storing values in list

```
//storing values for dangerous_enum_conversion_value
JsonObject dangerous_enum_conversion_value=
(JsonObject)bug_type.get("dangerous_enum_conversion");

inner_elements = new ArrayList<>();

for(String key : dangerous_enum_conversion_value.keySet()) {
      if(dangerous_enum_conversion_value.get(key).isJsonPrimitive()) {

            if(dangerous_enum_conversion_value.get(key).getAsJsonPrimi
            tive().isString()) {
                  str =
dangerous_enum_conversion_value.get(key).getAsString();
```

```
                        inner_elements.add(str);
                    } else {
                        bool =
dangerous_enum_conversion_value.get(key).getAsBoolean();
                        inner_elements.add(bool);
                    }
                } else {
                    array =
dangerous_enum_conversion_value.get(key).getAsJsonArray();
                        inner_elements.add(array);
                }

        }

                    jsonMap.put("dangerous{\\_}enum{\\_}conversion",
inner_elements);
```

- Coding the latex template and writing it to a TEX file

```
//Coding the latex template.
String latex = "\\documentclass{article} \r\n";
latex += "\\usepackage[utf8]{inputenc} \r\n";
latex += "\\usepackage{ffcode} \r\n";
latex += "\r\n";
latex += "\\title{Metasecurelabs analysis report}\r\n";
latex += "\\author{metasecurelabs.io }\r\n";
latex += "\\date{\\today} \r\n";
latex += "\r\n";
latex += "\\begin{document} \r\n";
latex += "\r\n";
latex += "\\maketitle \r\n";
latex += "\r\n";
latex += "\\section{Introduction} \r\n";

//Creating the TEX file
```

```
 File myObj = new File("json_data.tex");
 if (myObj.createNewFile()) {
      System.out.println("File created: " + myObj.getName());
} else {
      System.out.println("File already exists.");
}

 FileWriter myWriter = new FileWriter(myObj);

for(String key : jsonMap.keySet())
 {

       List<Object> inner = new ArrayList<>();
       inner = jsonMap.get(key);

       //Mapping keys and values from hashmap to the TEX file.
       latex += "\\subsection{" + key + "} \r\n";
       latex += "\\textbf{SWC{\\textunderscore }ID:}" + inner.get(5) +
"\r\n";
       latex += "\r\n";
       latex += "\\textbf{Description}:" + inner.get(1) + "\r\n";
       latex += "\r\n";
       latex += "\\textbf{Example:} \r\n";
       latex += "\\begin{ffcode} \r\n";
       latex += "\r\n";
       latex += inner.get(2);
       latex += "\r\n";
       latex += "\\end{ffcode} \r\n";
       latex += "\\} \r\n";
       latex += "\r\n";
       latex += "\\} \r\n";
       latex += "\r\n";
       latex += "\\textbf{DASP} : " + inner.get(0) + "\r\n";
       latex += "\r\n";
       latex += "\\textbf{Found}: " + inner.get(3) +  "\r\n";
       latex += "\r\n";
       latex += "\\textbf{Reported by checker}: \r\n";
       latex += "\\begin{ffcode} \r\n";
```

```
            latex += "\r\n";
            latex += inner.get(4);
            latex += "\r\n";
            latex += "\\end{ffcode} \r\n";
        }
            latex += "\\end{document}";

            //Writing the final latex template to the file created.
            myWriter.write(latex);
      System.out.println("Successfully wrote to the file.");

     //The file object is closed after it finishes writing.
            myWriter.close();
```

- Creating file objects and parsing special characters

```
      /*Reading from the initially created TEX file to parse special
characters which will be written to a newly created TEX File. The process
will be repeated each time we parse a set of special characters i.e if we
parse the file for one character, a new file A is created, if we parse the file
for another character, another file B is created etc */

      Reader fr = new FileReader(myObj);
      File tex = new File("latex.tex");
      FileWriter fw = new FileWriter(tex);
      BufferedReader br = new BufferedReader(fr);

      while(br.ready()) {
            fw.write(br.readLine().replace("_", "\\textunderscore ") +"\n");
      }

      fw.close();
      br.close();
      fr.close();
```

Complexities

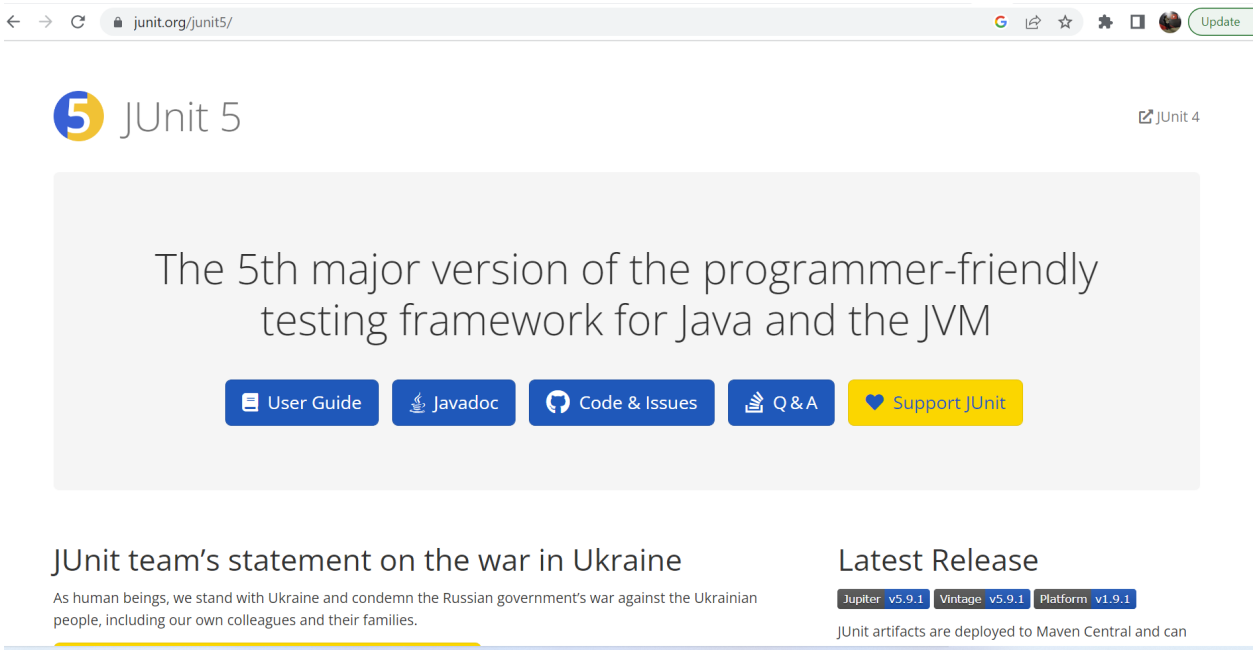
- The straightforward reason for utilizing a HashMap is performance. If we wish to find a specific element in a list, the time complexity is O(n), and if the list is sorted, it is O(log n) using a binary search.
- The advantage of using a HashMap is that the average time complexity to insert and retrieve a value is O(1). Because we're storing all of the specified elements in a map, the space complexity is O(n).

Risk Factors

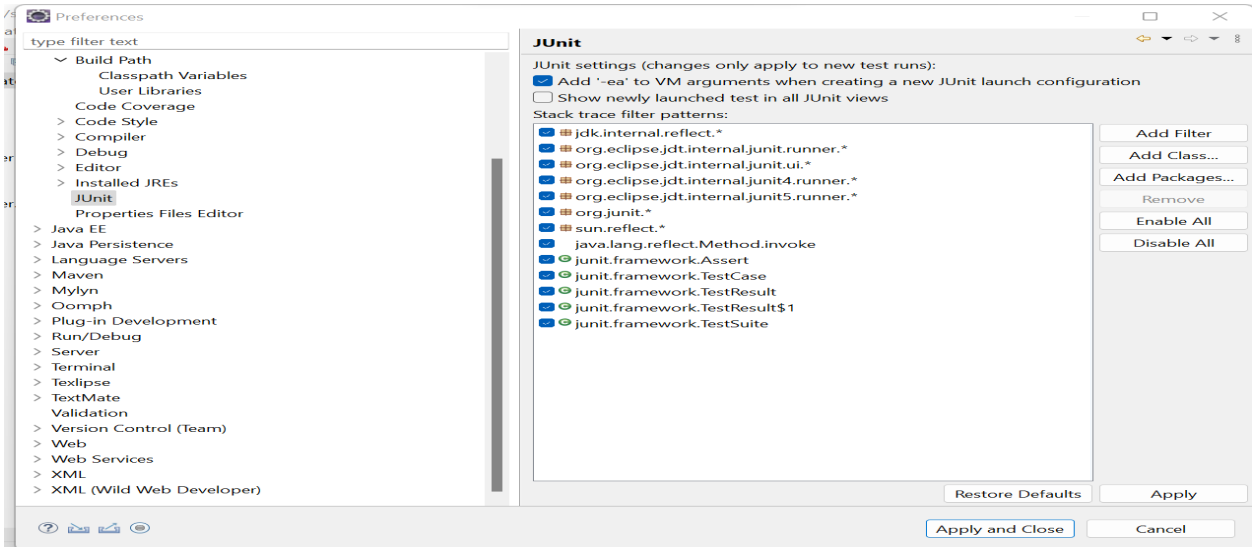
- If the newly created TEX file is not being found after the program has been run, refresh the project . It'll appear in the local directory of your project .
- When running pdflatex make sure it runs with -shell-escape .The -shell-escape allows cross engine interface execution i.e since we're using an external source for code format (ffcode). FFCODE for latex requires third party libraries for the code format. So to make it compatible, while compiling the TEX file, we use -shell-escape.
- While executing Junits, let's say if you're testing for no file exception or no element found , the tests should be within the try catch block else it'll throw an exception.

Appendix:

- Once the project has been imported, make sure **Junit5** is already installed, else install it from **https://junit.org/junit5/** .

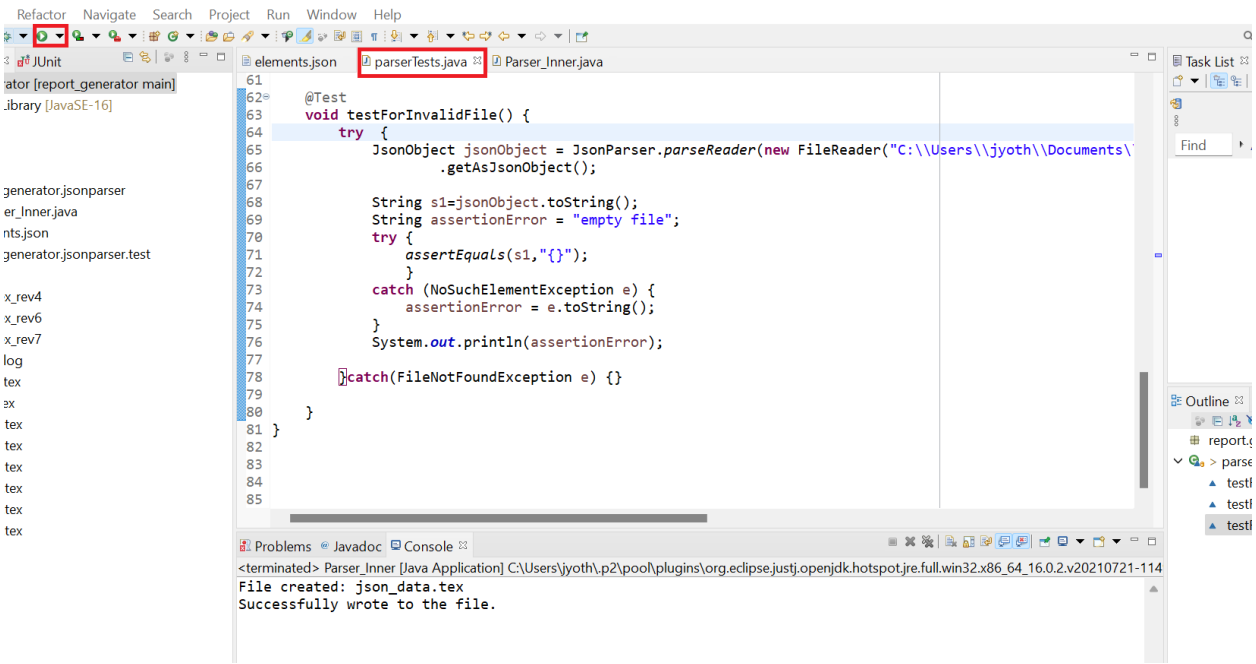


- Make sure the following checklist are checked as below:



- To include the **junit5 jar**, follow the steps similar to including **gson jar** as mentioned during the project setup.

- Once the **junit5 jar** has been included in your project, navigate to that test file and click run as shown below:



- The execution of tests looks green as shown below if all tests pass: