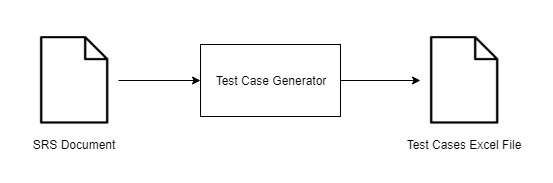
Documentation

Test Case Generator

What is it?

Test Case Generator is a web application that a user inserts an IEEE format SRS (Software Requirements Specification) document and get an Excel worksheet with all the chosen requirements in Test Case format.

Architecture:



This shows that the user has a SRS Document that he needs to extract the requirements from it, so when he drops the pdf file in the application and by following all the necessary steps, an Excel file with test cases will be generated. The next diagram will show, how the Test Case Generator component works in more detail:



The Test Case Generator component from above diagram has three components:

* Web Browser: Uses a HTML with the UI and a Javascript to manage the HTML’s buttons functionalities. This sub component uses two libraries: DropZone.js, used to create the drag and drop element on the web page. This library allows sending the dropped file to a specific URL or just to an automatic link, if the html file belongs to a Node.js server. The usage is very simple, which means that is very simple to configure and it comes with a good drag and drop CSS layout.

GOJS, used to create scenario’s flowcharts and make them more easily to understand by the user. It was chosen because it’s very customizable and works in a Web Browser (does not need Node.js to work and, because of that, it is possible to create dynamic flowcharts).

* JS Server: It is a Node.js server that works as a non-cache Proxy between the Web Browser and the Java server. About npm packages, this server uses: Express, that allows a http server creation. This package allows a very easy server creation. It is just needed to insert the specific URL and a function to handle the request and the response.

For a simple server it works fine, but when is it needed to send files, other packages are required.

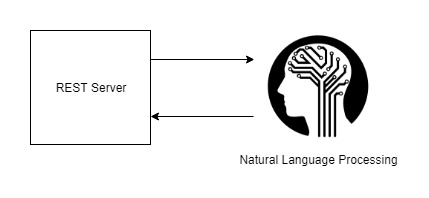
Connect-busboy, used as a file handler, which means that, when the server receives a file, this package will automatically recognize it and will write it into the server’s file folder. This write is used for persistence.

File-system, it is a simple way to read/write files and create streams in JavaScript. This package no longer supports asynchronous I/O operations.

Multer, it is a “multipart/form-data” parser. Used to interpret the FormData send it from the Web Browser with the selected Diagrams.

Request, used to create HTTP requests to JAVA server. A request’s content type is a “multipart/form-data” because was the only way found to send files via HTTP. Send it as a byte array, provokes in the other endpoint a file corruption. This error was caused by the sent file’s encoding. The only way found was send literally this file and do not convert it into a byte array. A form data is like a box that we can attach multiple data with multiple content type. The “box” was created and has the file that is going to be send inside. The other endpoint receives the file and the correct file’s content type too.

* Java Server: Inside this component there is another two sub components to consider:



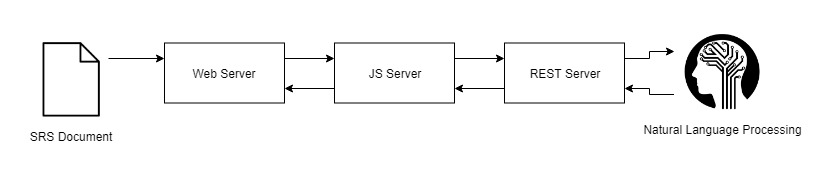
Rest Server, it is a simple server that uses REST to handle HTTP request. This server extracts the requirements from the SRS file, extracts test names and descriptions from an Excel file (not used yes, will be explained below) and writes the final Excel Test Case file.

This server has dependencies too. They are:

* iTextPDF, used for ready and extract the requirements page by page.
* Apache POI, used to read and write Microsoft type documents (in this case only Excel type was used)
* GSON, the Google’s JSON to Object and vice-versa converter
* COMMON-IO, used to simply load data into byte arrays
* Jersey, the API that allows the REST server creation. Was tried to use Jersey’s Form Data Handlers but unsuccessfully (unknown bug). Then was used a simple InputStream and Jersey’s FormDataParam annotation. The request handling was done manually.

Natural Language Processing, for this component it is used Monkey Learn. Monkey Learn it is a Natural Language Classifier API that is used to classify certain texts. In this case, it is used in requirements classification. This API was chosen because it is the only one found that is possible create a custom classification model for free. It is accessed via HTTP and receives a JSON for request and responses a JSON file format too.

Workflow:



Test Case Generator complete steps are:

* The user opens the web page
* Drops his SRS Document in drag and drop element
* The drag and drop, with the Dropzone.js, send the file automatically to the JS Server
* In JS Server, by using file-system package, the file is written in server’s files folder
* It is created a read file stream to get the file information and it is attached into a form data. This form data is sent to the Java Rest Server.
* The Rest Server reads the file and extracts the data form it.
* A JSON is created with all the found requirements and sent to the MonkeyLearn server to be classified by requirement type
* MonkeyLearn’s response comes in JSON format and it is sent to Rest Server
* Rest Server transforms the MonkeyLearn’s JSON into a customized JSON file
* This new customized JSON is sent as a response to JS Server and then passed through the web JavaScript
* The web JavaScript process this JSON and transforms each requirement found (only requirements with scenarios for now) into a flowchart. All diagrams are pushed into an array and the first one is popped out from this array and drawn to the diagram canvas
* By pressing the select buttons, the diagrams chosen are inserted into another FormData. When there are no more diagrams do show this FormData is sent to the JS Server
* On JS Server, using multer, the FormData is parsed and send it directly to REST Server
* On REST server, the FormData is transformed into Diagram Objects and inserted into an Excel File
* After the Excel File’s data insertion, it is written into a file
* This file is sent as Response from the REST Server
* The JS Server receives the file and it is automatically written into JS Server’s files folder. The path to this is sent to the Web Browser as response to the FormData sent it previously with the diagrams information
* When the path arrives to the Web JavaScript, it is created a new DOM download button in html. When it is pressed, the file is downloaded

Documentation:

All the developed code is commented, so for a good explanation about all the implemented functions, read the code’s comments.

What is missing?

This project for now, has a minimalist base. The idea was always

* Right now, the requirements used are only those that have scenarios so,