Semantic Data Web Technologies/EIS Lab

Summer Semester 2016

Technical Documentation

EULAide: Ontology-Driven Summarization and Visualization of Legal Documents

Project Team

Ahmad Hemid

Lars Möllenbrok

Isunge Mwangase

Md Saiful Islam Faisal

Project Mentor

Najmeh Mousavi Nejad

Table of Contents

1. Introduction

We all have to deal with end user license agreements (EULAs), while using computers. This happens when you install software or sign up for a web service and typically consist of several pages. But because of the length and the jargon of these documents most of us have gotten into the habit of simply accepting these agreements without looking at them which may lead to unexpected or unwanted consequences. In our project we tackle this problem by developing a system (EULAide) that uses ontology-based information extraction to quickly summarize a license, classify the different terms and conditions and visualize the results.

1.1 Purpose

This document provides a detailed overview of the software application. It is to be used in the software development process by stakeholders, developers, testers and project leaders.

1.2 Scope

The scope of the project is defined as follows:

- Extracting structures and semantics of vocabularies related to legal terms
- Summarize the legal document based on a predefined ontology
- Classify the sentences based on the semantics of vocabularies
- Present the output in an easy way to understand web GUI to the customer
- Work as a cross-platform software

1.3 Definitions, Acronyms and Abbreviations

API – Application Programming Interface

CSS – Cascading Style Sheets

EULA – End User License Agreement

FR – Functional Requirement

GUI – Graphic User Interface

HTML– Hypertext Mark-up Language

HTTP –Hypertext Transfer Protocol

ID – Identification

IE – Information Extraction

JDK – Java Development Kit

MVC – Model View Controller

NFR – Non-Functional Requirement

REST – Representational State Transfer

UI – User Interface

UML – Unified Modeling Language

URL – Uniform Resource Locator

2. The Overall Description

2.1 Product Perspective

A lot of people accept EULAs without even reading their content or understanding the impacts of accepting to the terms and conditions of such documents. From this vantage point, a software that can summarize the content of an EULA and clearly show the important legal terms it contains was developed. A few of recent research scientists have put effort on such a problem, a master thesis [1] related to the project has touched the extraction part of our project in the license domain. EULAide delivers a quick, scalable and fast interface which is highly needed, showing the legal terms contained in EULAs with their summaries.

2.2 Scenario

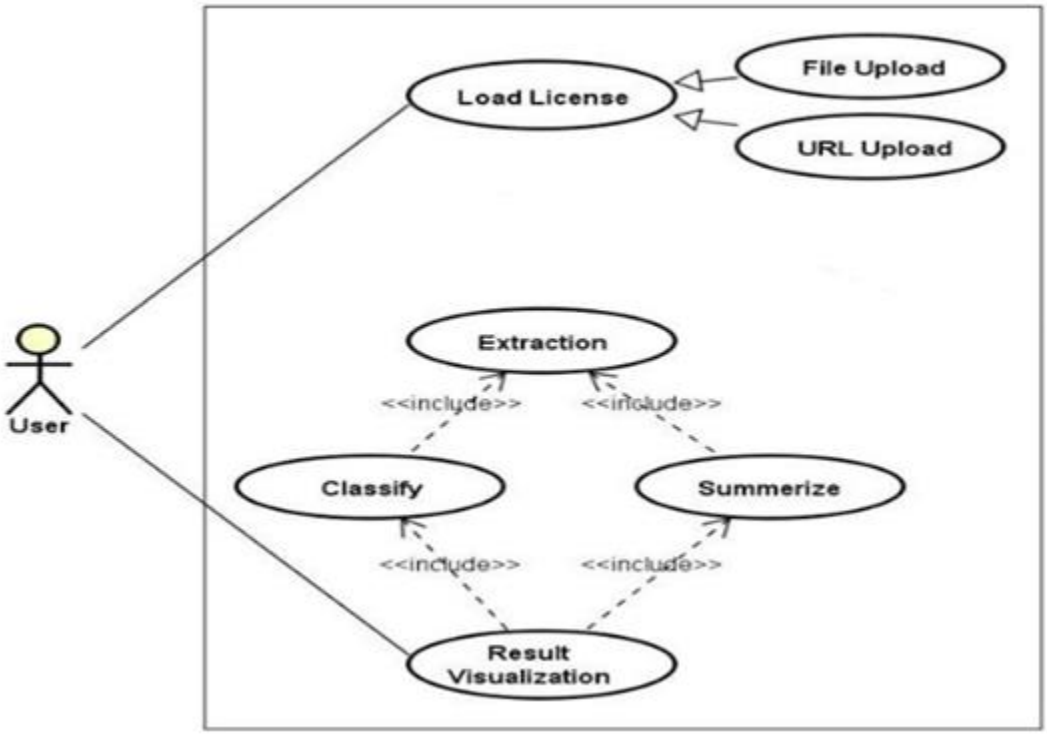


Figure 1: UML use case diagram

<u>Step by step explanation:</u>

Step 0: Preloaded Ontology

The system is using an existing or preloaded ontology, focused on legal documents (EULAs).

Step 1: User opens EULAide

User opens EULAide to summarize an EULA

Step 2: Load License

User can load an EULA with either the following ways:

- User uploads EULA .txt or .pdf file from local device or cloud
- User pastes link of EULA to EULAide

Step 3: Extraction

Legal terms are extracted and sematified using GATE pipeline

Step 4: Classification and Summarization

Different legal terms are classified into annotation sets (permission, duty, prohibition) and summarized

Step 5: Visualization

Semantified legal terms are visualized using icons for general users

2.3 User Characteristics

The system is implemented to support either general or advanced users. General user shall be anyone interested in knowing the contents of an EULA before and/or after installing software or signing up for an online service. Advanced users are those who are interested in finding out more details contained in the EULA.

2.4 Assumptions and Dependencies

- The ability to access the system with a remote access using internet or any kind of local connection between the server having the system and the user terminal
- The availability of a web browser on the user terminal
- JavaScript needs to be activated in the web browser

3. Architecture

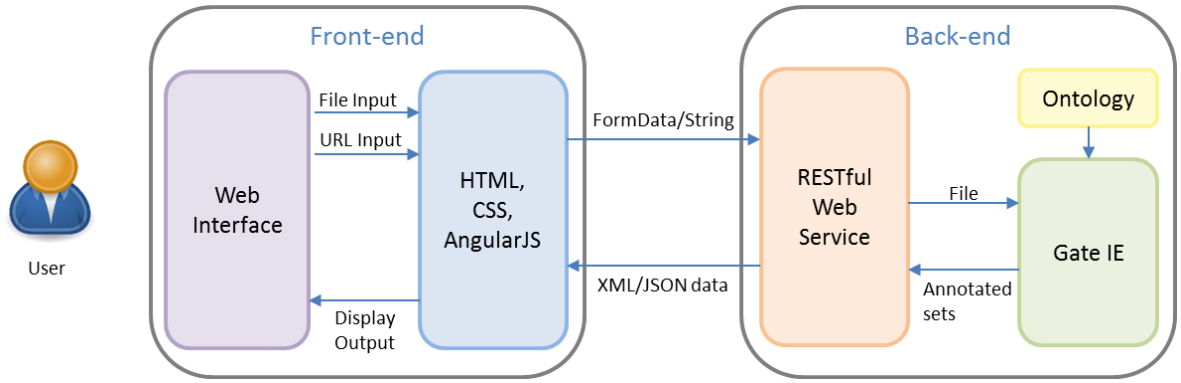


Figure 2: EULAide architecture

More details will be discussed in the implementation section.

4. Requirement Specifications

The analysis of project requirements led to discussion with the customer on how those are formalized and implemented.

4.1 External Interfaces

The system allows the input of an EULA either by providing a URL or uploading a file from a certain storage. The output will be a summary of the document represented in such a way that the user will be able to easily find facts contained in the EULA. In the case of an unsuccessful upload of the document, the user will be prompted with an error message and no further processing will be done.

4.2 Functional Requirements

The following requirements concern functions of the system. A priority notation has been used. This notation is a marker for requirements importance. Priorities 1 and 2 highlight attributes of high importance and low importance respectively. The attributes have been implemented in order of priority.

ID	NAME (and Describtion)	PRIORITY
FR_1	**Semantification** of an EULA • The user is able to upload an EULA from or a URL • The EULA is semantified using a predefined ontology	1
FR_2	**Summarization** • The EULA is summarized using ontology-based extraction	1
FR_3	**Annotation Classes** • This phase separates the Lookup annotation set which contains all the ontology-derived annotations.	1
FR_4	**Web GUI** • The system presents the result on a web GUI	1
FR_5	**Handle exceptions** • The system displays an error message when a failure occurred	1
FR_6	**Customized ontology** • The predefined ontology can be customized to support a wide range of legal documents' areas	2

FR_7	**Tool tips** • The system shall provide different classifications tool tips, quoting the corresponding paragraph of the original document	2
FR_8	**Visualization** • The result is visualized using simple icons to help the user to understand the content of the EULA	2

Table 1: Functional requirements

.

4.3 Non-functional Requirements

For the non-functional requirements the priority notion is as in 4.2.

ID	CATEGORY	DESCRIBTION	PRIORITY
NFR_1	**Performance**	Response time is less than one minute	1
NFR_2		Supports simultaneous users	1
NFR_3	**Availability**	Achieves 99% uptime	2
NFR_4	**Usability**	User interface is in simple English language	1
NFR_5		User can give feedback to enhance the quality	2
NFR_6	**Security**	System checks the uploaded file or URL for malicious code	2
NFR_7	**Interoperability**	System is provided as a web service to operate as being part of another system	1

Table 2: Non-functional requirements

5. Implementation

EULAide is a RESTful web service developed using AngularJS, Bootstrap and Java with GATE framework using two servers, Glassfish and Tomcat.

The implantation comprises a two-layer architecture; a web server to build the presentation layer using a dynamic web development tool AngularJS and the other layer hosting the application logic where Gate framework is processing the EULA and giving back the result to the web server. The communication between both servers is implemented using a Java REST service to have smooth communication between them.

Due to concerns that the application server would have a high overload when accessed by multiple users simultaneously, two servers were used thereby increasing the performance of the system and also save resources. The web server handles HTTP requests during the processing of the EULA by the application server, this in turn increases loose-coupling between the front-end and back-end module. It can help in deployment configuration (forming cluster and dictating some web servers)[3].

5.1 Development Tools

The development tools used to implement the project.

Tool (incl. version)	Usage
Back-end development tools	
GATE Framework v.8.1	Offers text engineering tools for natural language processing tasks
Java JDK v.8	Java Development Kit
Maven Central Repository	Used to install required libraries
Jersey v.2.4.1	Java REST library
Jsoup v.1.9.2	HTML parser library used for URL upload fuctionality
Front-end development tools	
UI Bootstrap v.3.3.7	Contains HTML and CSS design templates
AngularJS v.1.5.8	Open source JavaScript MVC framework for the web
Apache Tomcat v.9	Used as server for web application
Others	
Eclipse Jee Neon	Integrated Development Environment

Gate GUI	Used for understanding the working of Gate framework and checking the annotation set : permission, duty and prohibition
Firefox, Chrome, Safari	Used for testing and development

Table 3: Developement tools

5.2 Project Structure

The project is structured into two divisions separating client from server. The client part contains the front-end related files which are deployed using Tomcat server whereas the server part contains all the back-end files running on Glassfish server. Below are screenshots of the folder structure and a brief explanation of their functionalities in the table that follows.

5.2.1 Front-end file structure

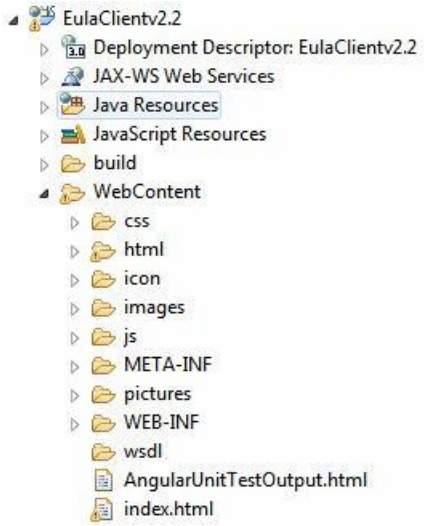


Figure 3: Front-end file structure

File/Folder/Package	Use
CSS	Contains CSS files for website design
HTML	Contains different HTML parts used in the website
IMAGES	Contains images required for visualization
JS	Contains JavaScript files (AngularJS)
PICTURES	Images used for the website
Index.html	The default page of the website

Table 4: File/Folder describtion

5.2.2 Back-end file structure

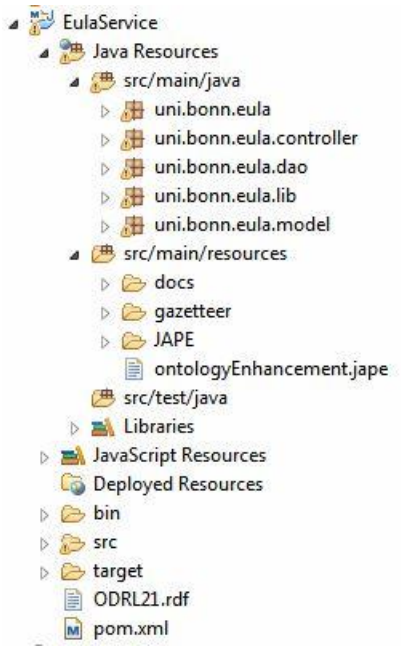


Figure 4: Back-end file structure

File/Folder/Package	Use
uni.bonn.eula	Contains main class to run back-end. Contains resource class for RESTful web service.
uni.bonn.eula.controller	For sets up GATE pipeline
uni.bonn.eula.dao	Helper Classes for RESTful service
uni.bonn.eula.model	Contains summary class
Docs	Contains EULAs that were used to test the service
Gazetter	Set of lists containing names of entities such as assets, countries, file formats, licenses name. They are used to find existences of these names in an EULA
JAPE	It provides finite state transduction over annotations based on regular expressions
ODRL21.rdf	An RDF file containing the default ontology to be preloaded in the system.
POM.xml	An XML file used by Maven to install the java libraries needed for the system.

Table 5: File/Folder/Package describtion

6. Testing

To ensure that the system is working without expected errors, different testing methods were performed. Follow as the testing methods we used in testing phase.

6.1 Unit Testing

In unit testing single components or modules of code are tested to make sure they are fit for use. In this project unit tests where focused on the front-end web logic, that is the AngularJS components. For testing Jasmine framework was used.

The following tests were done:

- File validation with right format (.txt, .pdf) and also with wrong format (.jpg or any other types) has been tested and succeded.
- URL validation with empty URL or non-empty URL has been tested.

Figure 5: Jasmine unit tests

6.2 Integration Testing

Integration testing is executed to make sure whether the components of a system interact with each other as defined in the requirement specifications or not. In RESTful API integration testing, the following need to be focused on:

- The HTTP response code

- Other HTTP headers in the response
- The payload (JSON, XML)

To perform this test, one of Google Chrome's API / web services testing tool called 'POSTMAN' was selected. It comes comes with powerful HTTP client support. The reasons for choosing this API for our integration test are,

- It has an easy-to-use request builder that allows writing of test cases, management of response data and response time for efficient testing and management of API test cases.
- Allows writing Boolean tests script within its interface.
- It tests run in a sandboxed environment, which is separated from the rest of the app [6,7,8].

Before we delve into testing, let's have a look at the resources available from our restful web service.

1. Root path - "/eula", then following paths for 4 available resources
    a. "/fileUploadXML"
    b. "/fileUploadJSON"
    c. "/urlUploadXML"
    d. "/urlUploadJSON"

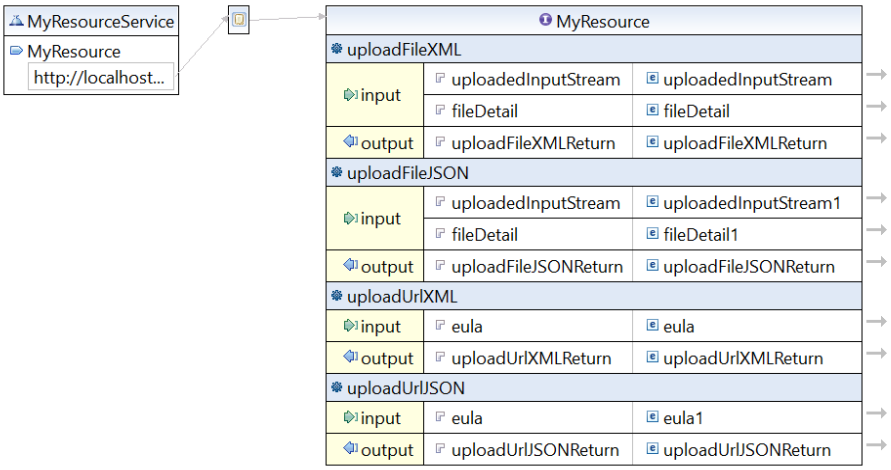


Table 6: Integration Test

- (a) and (b) take file input as [form-data] and return 'XML' and 'JSON' data respectively.
- On the other hand, (c) and (d) take URL input as [String] and returns 'XML' and 'JSON' respectively.

6.2.1 Testing script

A few lines of test scripts in Postman were used to get an automated test result.

For testing the status code of the response following script has been used:

```
tests["Status code is 200"] = responseCode.code === 200;
```

Then for testing if there was any content-type or any specific content-type ("***application/json***" / "***application/xml***") following script was used:

```
var contentTypeHeaderExists = responseHeaders.hasOwnProperty("Content-
Type");

tests["Has Content-Type"] = contentTypeHeaderExists;

if(contentTypeHeaderExists){
    tests["Content-Type is application/json"] = responseHeaders["Content-
Type"].has("application/json");
```

Now the crucial part of the response is the body content. In our case, there supposed to be three parts in the response, which are 'permission', 'duties' and 'prohibitions'

```
var jsonData = JSON.parse(responseBody);

tests["Response has permission"] = jsonData.hasOwnProperty("permissions");
tests["Response has duties"] = jsonData.hasOwnProperty("duties");
tests["Response has prohibitions"] =
```

annotation sets. For testing these annotation sets following scripts were used:

The scripts above were used to test the 'JSON' response while for testing 'XML' response the following scripts were used

```
tests["Body matches string = 'permissions'"] =
responseBody.has("permissions");
tests["Body matches string = 'duties'"] = responseBody.has("duties");
```

Last but not least, testing the time consuming by the services using the following script..

```
tests["Response time is less than 2500ms"] = responseTime < 2500;
```

6.2.2 Test Result

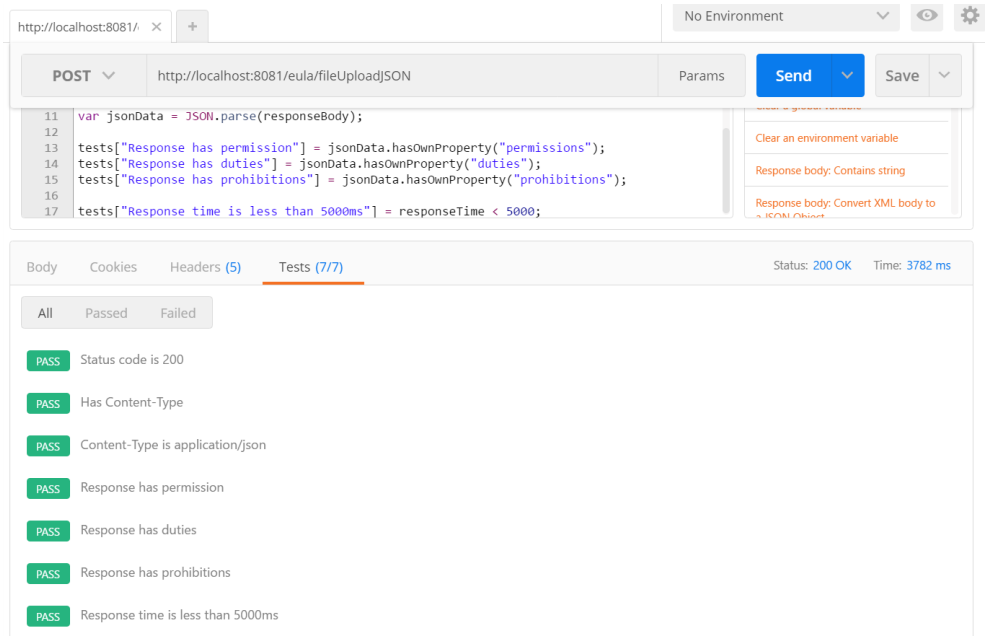


Figure 6: fileUploadJSON

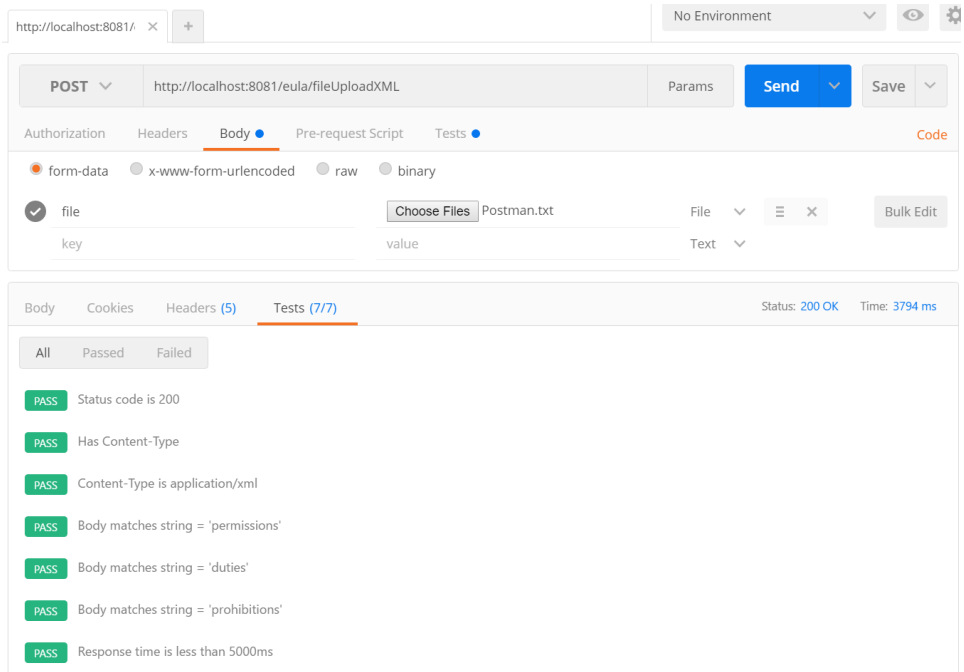


Figure 7: fileUploadXML

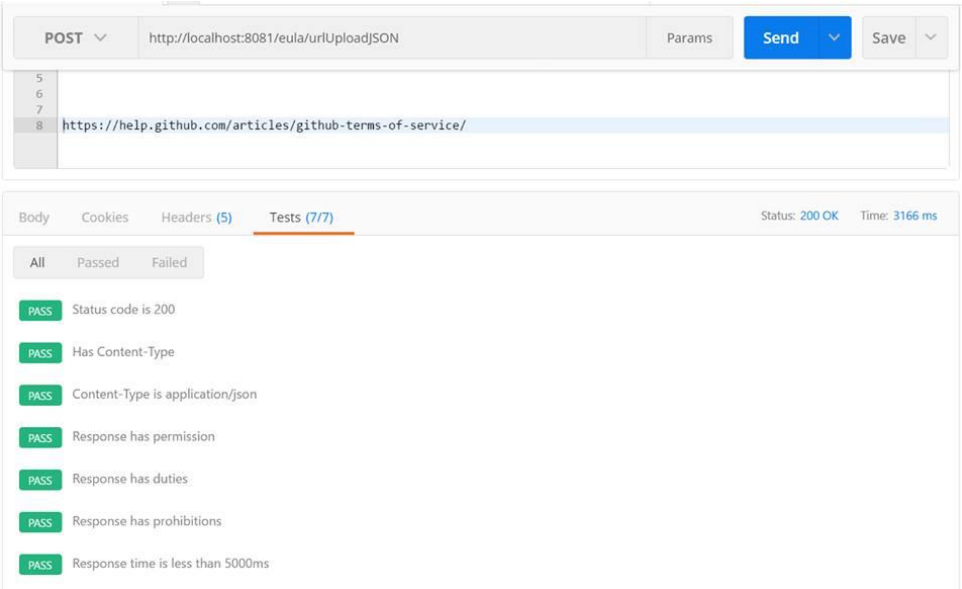


Figure 8: urlUploadJSON

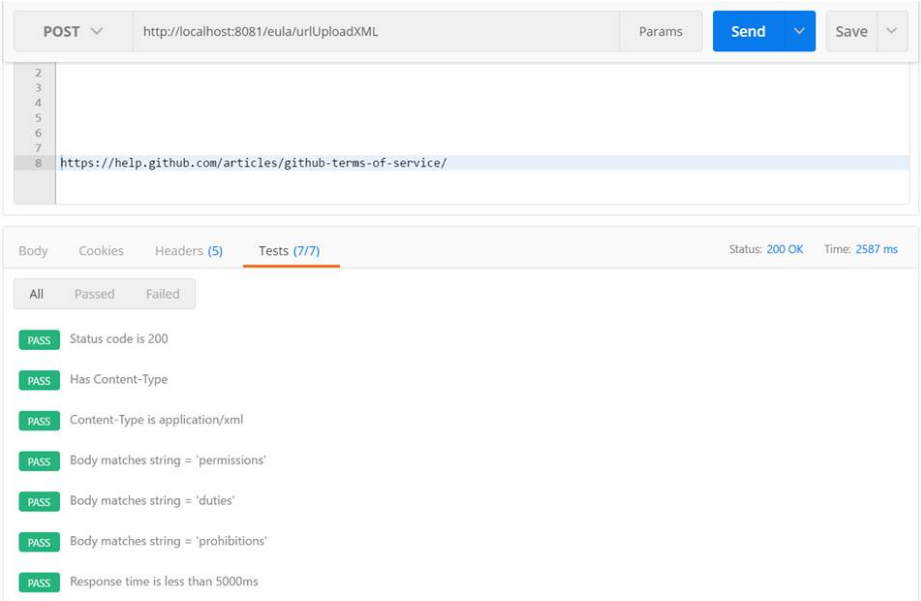


Figure 9:urlUploadXML

6.3 System Testing

In the preamble of this project, 20 well-known EULAs were provided to test EULAide from local disk. They produced desired outputs of annotation sets (prohibition, duties and permissions) successfully with a response time of less than five seconds (see pages 16/17 – Response time). In addition, the system was tested with ten EULAs of selected websites by providing the link of the EULA in the 'upload link' section of the application and yielded successful results with minor errors where one link did not produce desired results.

7. Mobile App Mockup

In addition to the EULAide web service, the system shall be implemented as a mobile application to allow a quick EULAide access for smartphone users. All major operating systems shall be supported. As the first step in the development process a mockup app was created. The full functional mockup can be downloaded at https://github.com/EIS-Bonn/MA-INF3232-4313-Lab-SS2016/tree/master/Gl/Dev.

Unlike the web service the EULAide mobile app will not offer generation from file/URL, but instead have an easy 'copy to'-functionality better fitting smartphone user behavior.

Figure 10: EULAide Mockuup

8. Future Work

- Implement the mockup in major mobile operating systems
- Offer different ontologies to cover various legal domains
- Availability in languages other than English

9. References

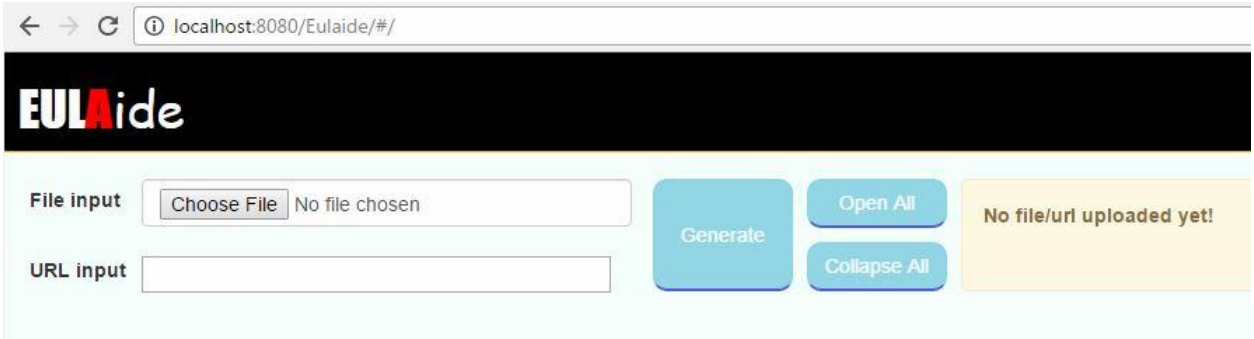
[1] Ken-Thomas Nilsen (2015). Ontology Based Information Extraction in the License Domain.
[2] Shzam Seshadri and Brad Green (2014) . AngularJS Up and Running
[3]http://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html
[4]http://www.agiletrailblazers.com/blog/6-reasons-to-use-postman/newman-for-api-integration-testing
[5]https://dzone.com/articles/12-great-web-service-testing-tools
[6]https://www.getpostman.com/docs/writing_tests

10. Appendix

10.1 User Manual

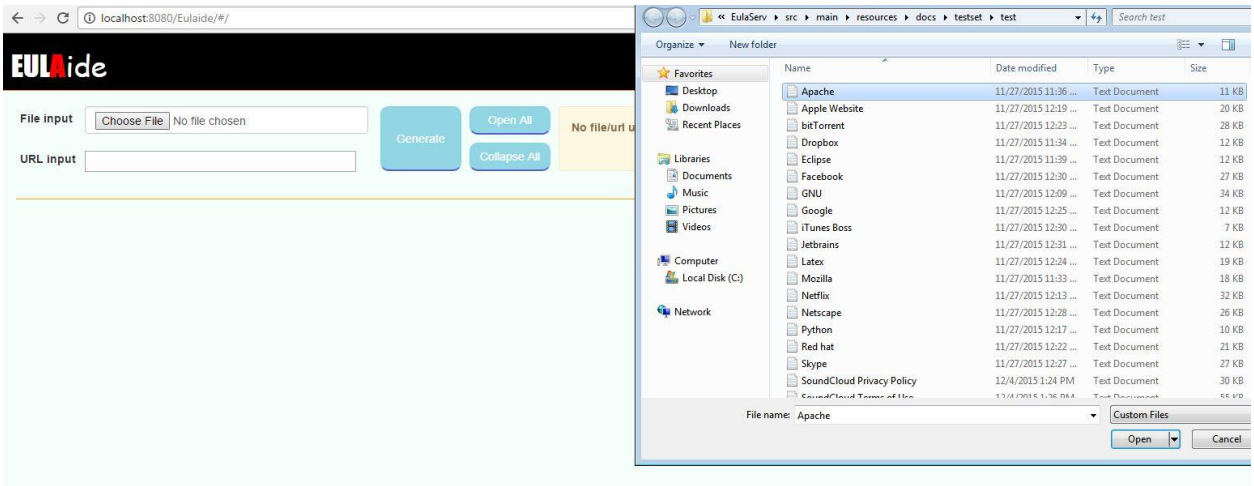
Steps to access EULAide

1- You need to access the website which is currently hosted under the following link: http://localhost:8080/Eulaide/#/
2- To submit the EULA, you have two choices, the first is to upload a text or pdf file, the second is to paste the URL of an EULA.



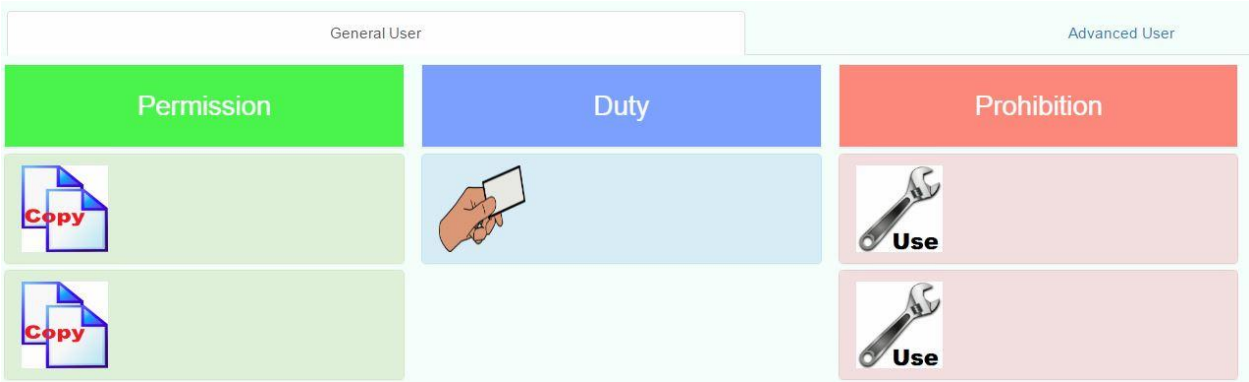
Uploading an EULA file

1- Click **Choose file** button and select the file according to the location of your system.
2- Click **Open** button

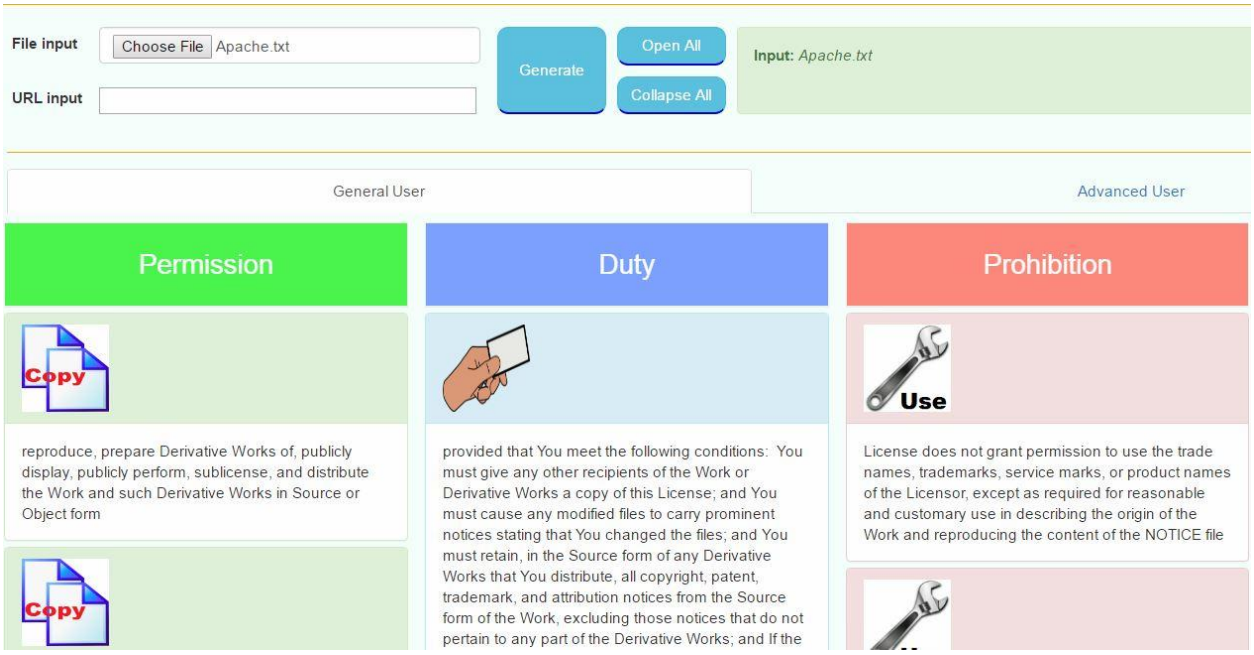


3- Click **Generate** button

4- You should be able to see the result of the processed file according to the following image



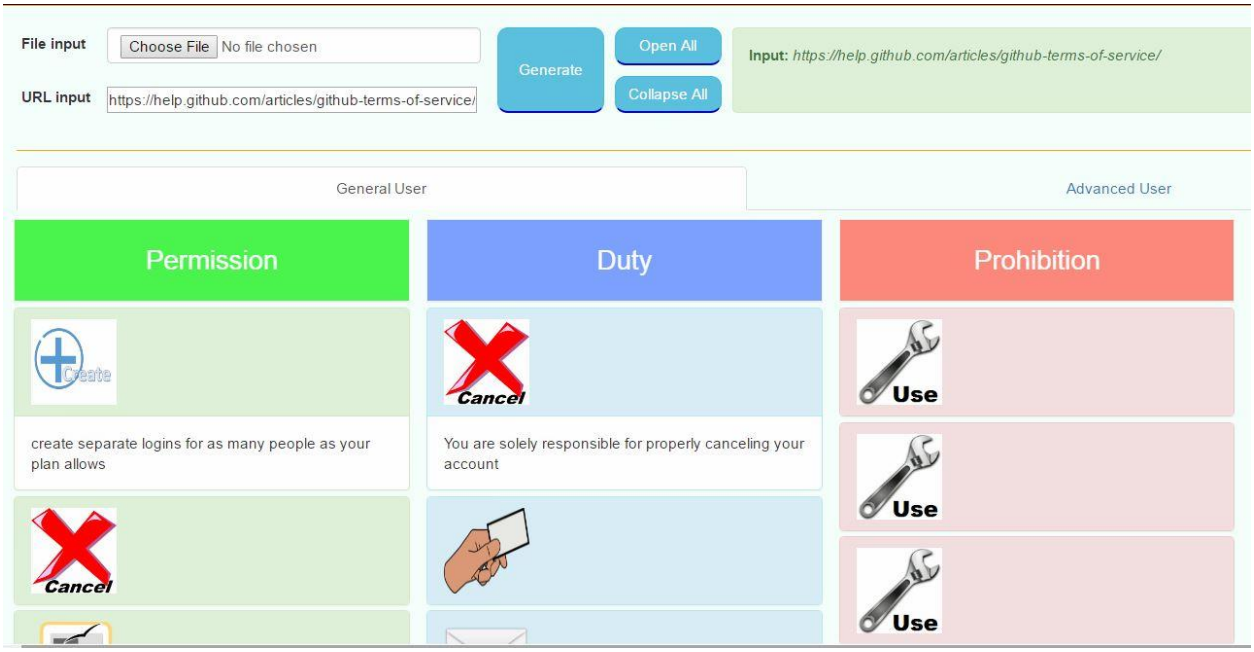
5- To expand single texts, click their icons. To expand all at once click **<u>Open All</u>** button. To hide all texts click **<u>Collapse All</u>** button.



6- For more details , click **Advanced User** button

Using the system by pasting an URL link

1- Copy the URL of an EULA you want to summarized
2- Paste the URL into the **URL input** box
3- Click **Generate** button
4- You should be able to see the result of the processed file according to the following image
5- To expand single texts, click their icons. To expand all at once click **Open All** button. To hide all texts click **Collapse All** button.
6- For more details , click **Advanced User** button



When using URL does not produce desired result

1- Check if URL is correct

2- If incorrect, correct it and try again

3- If the pasted URL still does not yield the desired result, copy the text content of the EULA of the URL and save it either as .txt or .pdf file and then use generation from file